Online Caching Networks with Adversarial Guarantees

YUANYUAN LI, Northeastern University, USA
TAREQ SI SALEM, Université Côte D'Azur, Inria, France
GIOVANNI NEGLIA, Inria, Université Côte D'Azur, France
STRATIS IOANNIDIS, Northeastern University, USA

We study a cache network under arbitrary adversarial request arrivals. We propose a distributed online policy based on the online tabular greedy algorithm [77]. Our distributed policy achieves sublinear $(1 - \frac{1}{e})$-regret, also in the case when update costs cannot be neglected. Numerical evaluation over several topologies supports our theoretical results and demonstrates that our algorithm outperforms state-of-art online cache algorithms.

CCS Concepts: • **Information systems → Storage management**; • **Theory of computation → Online algorithms**; **Caching and paging algorithms**; **Distributed algorithms**; *Submodular optimization and polymatroids.*

Additional Key Words and Phrases: Caching Network, Adversarial Guarantees, Online Optimization, No-Regret Algorithms

ACM Reference Format:
Yuanyuan Li, Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2021. Online Caching Networks with Adversarial Guarantees. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 3, Article 35 (December 2021), 39 pages. https://doi.org/10.1145/3491047

1 INTRODUCTION

We study network of caches, represented by an arbitrary topology, in which requests for contents arrive in an adversarial fashion. Requests follow paths along the network towards designated servers, that permanently store the requested items, and can be served by caches of finite capacity residing at intermediate nodes. Responses are carried back over the same path towards the source of each request, incurring a cost. Our objective is to propose a distributed, online algorithm determining cache contents in a way that minimizes regret, when both items requested as well as paths they follow are selected adversarially.

The offline version of this problem is NP-hard, but admits a $(1 - 1/e)$ polytime approximation algorithm [73]. Ioannidis and Yeh [41] proposed a distributed Robbins Monro type algorithm that attains the same approximation guarantee assuming stochastic, stationary request arrivals. This model has motivated several variants [35, 43, 54–57, 79], the majority of which focus on offline and/or stationary stochastic versions of the problem. Another thread of recent research in caching, spurred by the seminal work of Paschos et al. [66], explores caching algorithms that come with adversarial guarantees. The majority of these works focus either on a single cache [62, 71, 74] or on simple, bipartite network topologies [10, 65, 66].

Authors' addresses: Yuanyuan Li, yuanyuanli@ece.neu.edu, Northeastern University, Boston, USA; Tareq Si Salem, tareq.si-salem@inria.fr, Université Côte D'Azur, Inria, Sophia Antipolis, France; Giovanni Neglia, giovanni.neglia@inria.fr, Inria, Université Côte D'Azur, Sophia Antipolis, France; Stratis Ioannidis, ioannidis@ece.neu.edu, Northeastern University, Boston, USA.

2476-1249/2021/12-ART35 $15.00
https://doi.org/10.1145/3491047

The main objective of this paper is to bring adversarial guarantees to the general network model proposed by Ioannidis and Yeh. From a technical standpoint, this requires a significant technical departure from the no-regret caching settings studied by prior art [10, 62, 65, 66, 71, 74], both due to its distributed nature and the generality of the network topology. For example, our objective cannot be optimized directly via techniques from online convex optimization [39, 72, 81] to attain sublinear regret.

We make the following contributions:

- We revisit the general cache network setting of Ioannidis and Yeh [41] from an adversarial point of view.
- We propose DistributedTGOnline, a distributed, online algorithm that attains $O(\sqrt{T})$ regret with respect to an offline solution that is within a $(1-1/e)$-approximation from the optimal, when cache update costs are not taken into account.
- We also extend our algorithm to account for update costs. We show that an $O(\sqrt{T})$ regret is still attainable in this setting, replacing however independent caching decisions across rounds with coupled ones; we determine the latter by solving an optimal transport problem.
- Finally, we extensively evaluate the performance of our proposed algorithm against several competitors, using (both synthetic and trace-driven) experiments involving non-stationary demands.

The remainder of this paper is organized as follows. In Section 2, we review related work. Our model and distributed online algorithm are presented in Sections 3 and 4, respectively. We present our analysis of the regret under update costs in Section 5 and extend our results in Section 6. Our experiments in Section 7. We conclude in Section 8.

2 RELATED WORK

Content allocation in networks of caches has been explored in the offline setting, presuming demand is known [12, 68, 73]. In particular, Shanmugam et al. [73] were the first to observe that caching can be formulated as a submodular maximization problem under matroid constraints and prove its NP-hardness. Dynamic caching policies have been mostly investigated under a stochastic request process. One line of work relies on the characteristic time approximation [18, 27, 32, 46, 47] (often referred to as Che's approximation) to study existing caching policies [3, 7, 22, 31] and to design new policies that optimize the performance metric of interest (e.g., the hit ration or the average delay) [25, 53, 63]. Another line proposes caching policies inspired by Robbins-Monro/stochastic approximation algorithms [40, 41].

In particular, Ioannidis and Yeh [42] present (a) a projected gradient ascent (PGA) policy that attains $(1-1/e)$-approximation guarantee in expectation when requests are stationary and (b) a practical greedy path replication heuristic (GRD) that performs well in many cases, but comes without guarantee. Our work inherits all modeling assumptions on network operation and costs from [42], but differs from it (and all papers mentioned above) by considering requests that arrive adversarially. In our experiments, we compare our caching policy with PGA and GRD, that have no guarantees in the adversarial setting. We also prove that GRD in particular has linear regret (see Lemma 4.3).

Sleator and Tarjan [75] were the first to study caching under adversarial requests. In order to evaluate the quality of a caching policy, they introduced the competitive ratio, that is the ratio between the performance of the caching policy (usually expressed in terms of the miss ratio) and that of the optimal clairvoyant policy that knows the whole sequence of requests. This problem was generalized under the name of k-server problem [59] and metrical task system [11] and originated a vast literature (see, e.g., the survey [51]). In this paper, we focus on regret rather than competitive ratio to quantify the main performance metric. Roughly speaking, the regret corresponds to the difference between the performance of the caching policy and the optimal clairvoyant policy (see [4] for a thorough comparison of regret and competitive ratio). The regret metric is more popular in the online learning literature. The goal is to design algorithms whose regret grows sublinearly with the time horizon T and thus have asymptotically optimal time-average performance.

To the best of our knowledge, Paschos et al. [66] were the first to apply online learning techniques to caching. In particular, building on the online convex optimization framework [39, 72, 81], they propose an online gradient descent caching algorithm with sublinear regret guarantees, both for a single cache and for a bipartite network where users have access to a set of parallel caches (the "femtocaching" scenario in [73]) and items are random linearly encoded. Si Salem et al. [74] extend this work considering the more general family of online mirror descent algorithms [13], but only considered a single cache. Bhattacharjee et al. [10] prove tighter lower bounds for the regret in the femtocaching setting and proposed a caching policy based on the Follow-the-Perturbed-Leader algorithm that achieves near-optimal regret in the single cache setting. These results have been extended to the femtocaching setting [65]. Two recent papers [62, 71] pursued this line of work taking into account update costs for a single cache. We provide similar ($O(\sqrt{T})$) regret guarantees for general cache networks (rather than just bipartite ones), using a different algorithm.

As mentioned above, content placement at caches can be formulated as a submodular optimization problem [41, 73]. The offline problem is already NP-hard, but the greedy algorithm achieves $1/2$ approximation ratio [30]. Calinescu et al. [15] develop a $(1 - 1/e)$-approximation through the so called continuous greedy algorithm. The algorithm finds a maximum of the multilinear extension of the submodular objective using a Frank-Wolfe like gradient method. The solution is fractional and needs then to be rounded via pipage [2] or swap rounding [19]. Filmus and Ward [29] obtain the same approximation ratio without the need of a rounding procedure, by performing a non-oblivious local search starting from the solution of the usual greedy algorithm. These algorithms are suited for deterministic objective functions. Hassani et al. [38] study the problem of stochastic continuous submodular maximization and use stochastic gradient methods to reach a solution within a factor $1/2$ from the optimum. Mokhtari et al. [61] propose then the stochastic continuous greedy algorithm, which reduces the noise of gradient approximation by leveraging averaging technique. This algorithm closes the gap between stochastic and deterministic submodular problems achieving a $(1 - 1/e)$-approximation ratio.

There are two kinds of online submodular optimization problems. In the first one, a.k.a. competitive online setting, the elements in the ground set arrive one after the other, a setting considerably different from ours. The algorithm needs to decide whether to include revealed elements in the solution without knowing future arrivals. Gupta et al. [34] consider the case when this decision is irrevocable. They give a $O(\log r)$-competitive algorithm where r is the rank of matroid. Instead, Hubert Chan et al. [17] allow the algorithm also to remove elements from the current tentative solution. They propose a randomized 0.3178-competitive algorithm for partition matroids. In the second kind of online submodular optimization problems, objective functions are initially unknown and are progressively revealed over T rounds. This setting indeed corresponds to our problem, as our caching policy needs to decide the content allocation before seeing the requests. Streeter et al. [76] present an online greedy algorithm, combining the greedy algorithm with no-regret selection algorithm such as the hedge selector, operating under cardinality (rather than general matroid) constraints. Radlinski et al. [69] also propose an online algorithm by simulating the offline greedy algorithm, using a separate instance of the multi-armed bandit algorithm for each step of the greedy algorithm, also for cardinality constraints. Chen et al. [21] convert the offline Frank-Wolfe variant/continuous greedy to a no-regret online algorithm, obtaining a sublinear $(1 - 1/e)$-regret.

Chen et al. [20] use Stochastic Continuous Greedy [61] to achieve sublinear $(1-1/e)$-regret bound without projection and exact gradient. These algorithms however operate in a continuous domain, producing fractional solutions that require subsequent rounding steps; these rounding steps do not readily generalize to our distributed setting. Moreover, rounding issues are further exacerbated when needing to handle update costs in the regret.

Our work is based on the (centralized) TGOnline algorithm by Streeter et al. [77], which solves the so-called *online assignment problem.* In this problem, a fixed number of K slots is used to store items from distinct sets: that is, slot $k=1,\ldots,K$ can store items from a set P_k. The motivation comes, from, e.g., placing advertisements in K distinct positions on a website. Submodular reward functions arrive in an online fashion, and the goal of the online assignment problem is to produce assignments of items to slots that attain low regret. TGOnline, the algorithm proposed by Streeter et al., achieves sublinear $1-1/e$-regret in this setting. We depart from Streeter et al. by considering both objectives as well as constraints arising from the cache network design problem. We show that (a) when applied to this problem, TGOnline admits a distributed implementation, but also (b) we incorporate update costs, which are not considered by Streeter et al. A direct, naïve implementation of TGOnline to our setting would require communication between *all* caches at *every* request; in contrast, DistributedTGOnline restricts communication only among nodes on the request path. As an additional technical aside, we exploit the fact that an adaptation step within the TGOnline algorithm, namely, color shuffling, can in fact happen at a reduced frequency. The latter is imperative for bounding regret when cost updates are considered: without this adjustment, the TGOnline algorithm of Streeter et al. attains a linear regret when incorporating update costs.

3 MODEL

Following the caching network model of Ioannidis and Yeh [41], we consider a network of caches that store items from a fixed catalog. Nodes in the network generate requests for these items, routed towards designated servers. However, intermediate nodes can cache items and, thereby, serve such requests early. We depart from Ioannidis and Yeh in assuming that request arrivals are adversarial, rather than stochastic.

3.1 Notation

We use notation $[n] \triangleq \{1,2,...,n\}$ for sets of consecutive integers, and $\mathbb{1}(\cdot)$ for the indicator function, that equals 1 when its argument is true, and 0 otherwise. Given two sets A, B, we use $A\times B = \{(a,b)\}_{a\in A, b\in B}$ to indicate their Cartesian product. For any finite set A, we denote by $|A|\in\mathbb{N}$ the size of the set. For a set A and an element a, we use $A+a$ to indicate $A\cup\{a\}$. Notation used across the paper is summarized in Table 1.

3.2 Caching Network

We model a caching network as a directed graph $G(V,E)$ of n nodes. For convenience, we set $V=[n]$. Each edge e in the graph is represented by $e=(u,v)\in E\subseteq V\times V$. We assume G is symmetric, i.e., if $(u,v)\in E$, then $(v,u)\in E$. A fixed set of nodes, called designated servers, permanently store items of equal size. Formally, each item $i\in C$, where set C is the item catalog, is stored in designated servers $\mathcal{D}_i\subseteq V$.

Beyond designated servers, all other nodes in V are also capable of storing items. For each $v\in V$, let $c_v\in\mathbb{N}$ denote its storage capacity, i.e., the maximum number of items it can store. Let also $\mathcal{S}_v=\{(v,j)\}_{j=1}^{c_v}$ be v's set of storage slots; then, $s=(v,j)\in V\times[c_v]$ is the j-th storage slot on node v. We denote the set of storage slots in the whole network by $\mathcal{S}$, where $\mathcal{S}=\bigcup_{v\in V}\mathcal{S}_v$, and

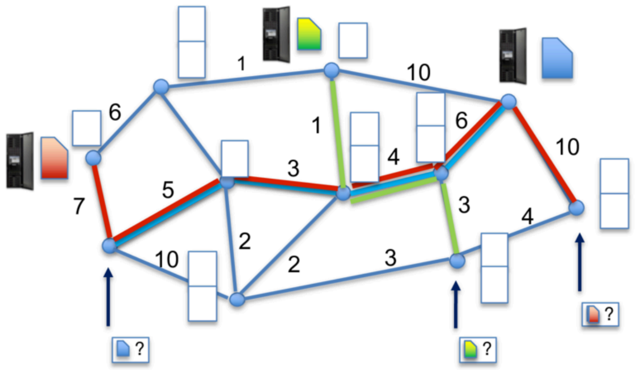


Fig. 1. A general cache network, as proposed by Ioannidis and Yeh [41]. Designated servers store items in a catalog permanently. Requests arrive at arbitrary nodes in the network and follow predetermined paths towards these servers; responses incur costs indicated by weights on the edges. Intermediate nodes can serve as caches; the objective is to determine what items to store in each cache, to minimize the overall routing cost or, equivalently, maximize the caching gain.

$|\mathcal{S}| = \sum_{v \in V} c_v$. We assume the slots in $\mathcal{S}$ are ordered lexicographically, i.e.,

$$(v, j) \prec (v', j') \text{ if and only if } v < v' \text{ or } v = v' \text{ and } j < j'. \tag{1}$$

We can describe content allocation as a set $A \subset \mathcal{S} \times C$, where $a = (s, i) \in A$ indicates that item $i \in C$ is stored in slot $s \in \mathcal{S}$. The set of feasible allocations is

$$\mathcal{D} = \{A \subseteq \mathcal{S} \times C : |A \cap (\{s\} \times C)| \leq 1, \forall s \in \mathcal{S}\}. \tag{2}$$

This ensures that each slot is occupied by at most one item; note that the cache capacity constraint at each node $v \in V$ is captured by the definition of $\mathcal{S}_v$.

3.3 Requests and Responses

A request $r = (i, p)$ is determined by (a) the item $i \in C$ requested, (b) the path p along which the request is forwarded. A path p is a sequence $\{p_k\}_{k=1}^{|p|}$ of adjacent nodes $p_k \in V$. As in Ioannidis and Yeh [41], we assume that paths are simple, i.e., they do not contain repeated nodes, and well-routed, i.e., they terminate at a node in $\mathcal{D}_i$. A request (i, p) is generated at node p_1 and follows the path p; when the request reaches a node storing item i, a response is generated. This response carries item i to query node p_1 following the reverse path. We assume that time is slotted, and requests arrive over a total of $T \in \mathbb{N}$ rounds. We denote by $\mathcal{R}$ the set of all possible requests in the system. At each round $t \in [T]$, a set of requests $\mathcal{R}^t \subseteq \mathcal{R}$ arrive in the system. Requests in $\mathcal{R}^t$ can arrive in any order, and at any point in time within a round.[1] However, we assume that the total number of requests at each round is bounded by $\bar{R}$, i.e., $|\mathcal{R}^t| \leq \bar{R}$. Note that, when $\bar{R} = 1$, at most one request arrives per round.

3.4 Routing Costs

We assume request routing does not incur any cost, but response routing does. In particular let $w_{uv} \in \mathbb{R}_+$ denote the cost of routing the response along the edge $(u, v) \in E$.

[1] Our analysis readily extends to a multiset $\mathcal{R}^t$, whereby the same request is submitted multiple times within the same round. We restrict the exposition to sets for notational simplicity.

Table 1. Notation Summary

	Notational Conventions
$[n]$	Set $\{1, \dots, n\}$
$A + a$	Union $A \cup \{a\}$
	Cache Networks
$G(V, E)$	Network graph, with nodes V and edges E
$\mathcal{C}$	The item catalog
c_v	Cache capacity at node $c \in V$
$s = (v, j)$	j-th storage slot on node v
$\mathcal{S}$	The set of storage slots
$\mathcal{S}_v$	The set of storage slots in node v
$\mathcal{S}_p$	The set of storage slots in path p
$\mathcal{S}_{i,p}$	The set of storage slots in path p storing item i
A	The set of item allocations
$\mathcal{D}$	The set of feasible allocations
$\mathcal{R}_t$	The set of requests arriving at round t
$\bar{R}$	The upper bound of $\lvert\mathcal{R}_t\rvert$
$\overline{\lvert\mathcal{R}_t\rvert}$	Average number of requests per round
w_{uv}	The routing cost along edge (u, v)
$\bar{L}$	The upper bound of possible routing cost
f^t	Caching gain at round t
	Online Optimization
T	The rounds horizon
R_T	α-regret
$\mathcal{E}$	Hedge selector
$\boldsymbol{W}$	The weight vector maintained by hedge selector
$\boldsymbol{\ell}$	The reward vector fed to hedge selector
m_s	The active color of slot s
M	Number of colors
$\mathcal{I}$	Information collected by control message
w_v^p	The cumulative cost of edges upstream of v on path p
UC	Update costs
$\tilde{R}_T$	The extended α-regret considering update costs

Then, given an allocation $A \in \mathcal{S} \times \mathcal{C}$, the cost of serving a request $(i, p) \in \mathcal{R}$ is:

$$C_{(i,p)}(A) = \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(A \cap \left\{ \bigcup_{k' \in [k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} = \emptyset \right). \tag{3}$$

Intuitively, Eq. (3) states that $C_{(i,p)}(A)$ includes $w_{p_{k+1}p_k}$, the cost of edge (p_{k+1}, p_k), only if no cache preceding p_{k+1} in path p stores item i. We denote by

$$\bar{L} = \max_{(i,p) \in \mathcal{R}} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \tag{4}$$

the maximum possible routing cost; note that this upper-bounds $C_{(i,p)}(A)$, for all $(i, p) \in \mathcal{R}$, $A \in \mathcal{S} \times \mathcal{C}$.

The *caching gain* [41] of a request (i, p) due to caching at intermediate nodes is:

$$f_{(i,p)}(A) = C_{(i,p)}(\emptyset) - C_{(i,p)}(A) = \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(A \cap \left\{ \bigcup_{k' \in [k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} \neq \emptyset \right). \tag{5}$$

where $C_{(i,p)}(\emptyset)$ is the routing cost in the network when all caches are empty. The caching gain captures the cost reduction in the network due to caching allocation A.

3.5 Offline Problem

In the offline version of the cache gain maximization problem [41, 73], the request sequence $\{\mathcal{R}^t\}_{t=1}^T$ is assumed to be known in advance; the goal is then to determine a feasible allocation $A \in \mathcal{D}$ that maximizes the total caching gain. Formally, given an allocation $A \in \mathcal{S} \times C$, let

$$f^t(A) = \sum_{r \in \mathcal{R}^t} f_r(A), \tag{6}$$

be the caching gain at round t. Then, the offline caching gain maximization problem amounts to:

$$\underset{A}{\text{maximize}} \quad f(A) = \sum_{t=1}^{T} f^t(A) = \sum_{t=1}^{T} \sum_{r \in \mathcal{R}^t} f_r(A), \tag{7a}$$

$$\text{subject to} \quad A \in \mathcal{D}. \tag{7b}$$

The following lemma implies this problem is approximable in polynomial time:

Lemma 3.1 ([41, 73]). *Function $f : \mathcal{S} \times C \to \mathbb{R}_+$ is non-decreasing and submodular. Moreover, the feasible region $\mathcal{D}$ is a partition matroid.*

Hence, Problem (7) is a submodular maximization problem under matroid constraints. It is known to be NP-hard [41, 73], and several approximation algorithms with polynomial time complexity exist. The classic greedy algorithm [14] produces a solution within $\frac{1}{2}$-approximation from the optimal. The so-called continuous greedy algorithm [15] further improves this ratio to $1 - \frac{1}{e}$. A different algorithm based on a convex relaxation of Problem (7) is presented in [41, 73]. The Tabular Greedy algorithm [77] also constructs a $1 - \frac{1}{e}$ approximate solution in poly-time. We describe it in details in Appendix A, as both TGOnline [77] and our DistributedTGOnline build on it.

3.6 Online Problem

In the online setting, requests are not known in advance, and we seek algorithms that make caching decisions in an online fashion. In particular, at the beginning of round t, an online algorithm selects the current allocation $A^t \in \mathcal{D}$. Requests $\mathcal{R}^t \subset \mathcal{R}$ subsequently arrive, and the cache gain $f^t(A^t)$ is rewarded, where $f^t : \mathcal{S} \times C \to \mathbb{R}_+$ is given by Eq. (6).

As in standard online learning literature [39, 66], while choosing A^t, the network has no knowledge of the upcoming requests $\mathcal{R}^t$, but can rely on past history. Formally, we seek an online algorithm $\mathcal{A}$ that maps the history of past requests $\mathcal{H}^t = \{\mathcal{R}^1, .., \mathcal{R}^{t-1}\}$ to a new allocation, i.e., $A^t = \mathcal{A}(\mathcal{H}^t)$. In particular, we aim for an algorithm $\mathcal{A}$ with sublinear α-regret, given by

$$R_T = \mathbb{E}\left[\alpha \sum_{t=1}^{T} f^t(A^*) - \sum_{t=1}^{T} f^t(A^t) \right] = \alpha \sum_{t=1}^{T} f^t(A^*) - \mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t) \right], \tag{8}$$

where α is an approximation factor, and A^* is the optimal solution to (the offline) Problem (7). Note that the expectation is over the (possibly) randomized choices of the algorithm $\mathcal{A}$; we make no probabilistic assumptions on request arrivals $\{\mathcal{R}^t\}_{t=1}^T$, and wish to minimize regret in the adversarial setting, i.e., w.r.t. to the worst-case sequence $\{\mathcal{R}^t\}_{t=1}^T$. Put differently, our regret bounds will be

Algorithm 1: Hedge Selector $\mathcal{E}$

```
Input: Parameter $\epsilon \in \mathbb{R}_+$, action set $C$, horizon $T \in \mathbb{N}$.
def $\mathcal{E}$.initialize():
    Set $W_i \leftarrow 1$ for all $i \in C$
def $\mathcal{E}$.arm():
    return $i \in C$ with probability $p_i = \frac{W_i}{\sum_{j \in C} W_j}$
def $\mathcal{E}$.feedback($\boldsymbol{\ell}$):
    Set $W_i \leftarrow W_i e^{\epsilon \ell_i}$ for all $i \in C$
```

against an arbitrarily powerful adversary, that can pick *any* sequence $\{\mathcal{R}^t\}_{t=1}^T$, as long as the total number of requests at each round is bounded by $\bar{R}$, i.e., $|\mathcal{R}^t| \leq \bar{R}$ for all $t = 1, \ldots, T$.

Several remarks are in order regarding Eq. (8). First, the definition of the regret in Eq. (8), which compares to a static offline solution, is classic. Several bandit settings, e.g., simple multi-armed bandits [6, 50, 69], contextual bandits [1, 23, 26], submodular bandits [21, 76, 80] and, of course, their applications to caching problems [10, 62, 66, 71], adopt this definition. In all these cases, the dynamic, adaptive algorithm is compared to a static policy that has full hindsight of the entire trace of actions. Nevertheless, as is customary in the context of online problems in which the offline problem is NP-hard [21], the regret is not w.r.t. the optimal caching gain, but the gain obtained by an offline approximation algorithm. Second, from the point of view of bandits, we operate in the full-information feedback setting [39]: upon the arrival of requests $\mathcal{R}^t$, the entire function $f^t : \mathcal{S} \times C \to \mathbb{R}_+$ is revealed,[2] as the latter is fully determined by request set $\mathcal{R}^t$. Third, Eq. (8) captures the cost of serving requests, but not the cost of adaptation: changing an allocation from A^t to A^{t+1} changes cache content, which in turn may require the movement of items. Neglecting adaptation costs may be realistic if, e.g., adaptation happens in off-peak hours (e.g., the end of a round occurs at the end of a day), and does not come with the same latency requirements as serving requests in $\mathcal{R}^t$. Nevertheless, we revisit this issue, incorporating update costs in the regret, in Section 5. Finally, we stress that we seek online algorithms $\mathcal{A}$ that have sublinear regret but are also distributed: each cache v should be able to determine its own contents using past history it has observed, as well as some limited information it exchanges with other nodes.

4 DISTRIBUTED ONLINE ALGORITHM

We describe our distributed online algorithm, DistributedTGOnline, in this section. We first give an overview of the algorithm and its adversarial guarantees; we then fill out missing implementation details.

4.1 Hedge Selector

Our construction uses as a building block the classic Hedge algorithm[3] for the expert advice problem [5, 39, 77]. This online algorithm selects an action from a finite set at the beginning of a round. At the conclusion of a round, the rewards of all actions are revealed; the algorithm accrues a reward based on the action initially selected, and adjusts its decision.

In our case the set of possible actions coincides with the catalog C, i.e., the algorithm selects an item $i^t \in C$ per round $t \in \mathbb{N}$. The hedge selector maintains a weight vector $\boldsymbol{W}^t = [W_i^t]_{i \in C} \in \mathbb{R}^{|C|}$, where weight W_i^t corresponds to action $i \in C$. The hedge selector supports two operations (see

[2] In contrast to the classic bandit feedback model, where only the reward $f^t(A^t)$ is revealed in each round t.

[3] This is also known as the multiplicative weight algorithm.

Algorithm 2: DistributedTGOnline

```
foreach $s \in \mathcal{S}$ do
    foreach $m \in [M]$ do
        $\mathcal{E}_{s,m}$.initialize().
    set $t_s = 1$, choose $m_s$ uniformly at random from $[M]$
    $i_s \leftarrow \mathcal{E}_{s,m_s}$.arm() ;                        // Sets $A \leftarrow \{(s, i_s)\}_{s \in \mathcal{S}}$
for $t = 1, 2, \dots, T$ do
    /* During round $t$:                                                */
    foreach $r = (i, p) \in \mathcal{R}^t$ do
        Send request upstream over $p$ until a hit occurs.
        Send response carrying $i$ downstream, and incur routing costs.
        Send control message upstream over $p$ to construct $\mathcal{I}$ and $\mathcal{W}$, given by (12).
        Send control message carrying $\mathcal{I}$ and $\mathcal{W}$ downstream over $p$, and do the
          following:foreach $s \in \mathcal{S}_p$ do
            Use $\mathcal{I}$ and $\mathcal{W}$ to construct $\boldsymbol{\ell}(s, m_s) \in \mathbb{R}_+^{|C|}$ via (15).
            Call $\mathcal{E}_{s,m_s}$.feedback($\boldsymbol{\ell}(s, m_s)$)
    /* At the end of round $t$:                                         */
    foreach $s \in \bigcup_{(i,p) \in \mathcal{R}^t} \mathcal{S}_p$ do
        if $t_s \bmod K = 0$ then
            Select $m_s$ u.a.r. from $[M]$ ;                          // Shuffle color $m_s$
        $i_s \leftarrow \mathcal{E}_{s,m_s}$.arm() ;                      // Update allocation $A$ at $s$
        $t_s \leftarrow t_s + 1$
```

Alg. 1. The first, $\mathcal{E}$.arm( ), selects an action from action set C. The second, $\mathcal{E}$.feedback($\boldsymbol{\ell}^t$), ingests the reward vector $\boldsymbol{\ell}^t = [\ell_i^t]_{i \in C} \in \mathbb{R}_+^{|C|}$, where ℓ_i^t is the reward for choosing action $i \in C$ at round t, and adjusts action weights as described below. In each iteration t, the hedge selector alternates between (a) calling $i^t = \mathcal{E}$.arm( ), to produce action i^t, (b) receiving a reward vector $\boldsymbol{\ell}^t$, and using it via $\mathcal{E}$.feedback($\boldsymbol{\ell}^t$) to adjust its internal weight vector. In particular, $\mathcal{E}$.arm( ) selects action $i \in C$ with probability:

$$p_i^t = \frac{W_i^t}{\sum_{j \in C} W_j^t}, \tag{9}$$

i.e., proportionally to weight W_i^t. Moreover, when $\mathcal{E}$.feedback($\boldsymbol{\ell}^t$) is called, weights are updated via:

$$W_i^{t+1} = W_i^t e^{\epsilon \ell_i^t}, \quad \text{for all } i \in C, \tag{10}$$

where $\epsilon > 0$ is a constant. In a centralized setting where an adversary selects the vector of weights $\boldsymbol{\ell}^t$, the no-regret hedge selector attains an $O(\sqrt{T})$ regret for an appropriate choice of $\epsilon > 0$ (see Lemma B.1 in Appendix B). We use this as a building block in our construction below.

4.2 DistributedTGOnline Overview

To present the DistributedTGOnline algorithm, we first need to introduce the notion of "colors". The algorithm associates each storage slot $s = (v, j) \in \mathcal{S}$ with a "color" m_s from set $[M]$ of M distinct values (the "color palette"). The online algorithm makes selections in the extended action space $\mathcal{S} \times C \times [M]$, choosing not only an item to place in a slot, but also how to color it.

This coloring is instrumental to attaining a $1-\frac{1}{e}$-regret. In offline tabular greedy algorithm of Streeter et al. [77], which we present in Appendix A, when $M=1$, i.e., there is only one color, the algorithm reduces to simply the locally greedy algorithm (see Section 3.1 in [77]), achieving only a $\frac{1}{2}$ approximation ratio. When $M \to \infty$, the algorithm can intuitively be viewed as solving a continuous extension of the problem followed by a rounding (via the selection of the color instance), in the same spirit as the so-called ContinuousGreedy algorithm [15]. A finite size "color palette" represents a midpoint between these two extremes. We give more insight into this relationship between these two algorithms in Section 4.5.

In more detail, every storage slot $s \in \mathcal{S}$ maintains (a) the item $i_s \in C$ stored in it, (b) an active color $m_s \in [M]$ associated with this slot, and (c) M different no-regret hedge selectors $\{\mathcal{E}_{s,m}\}_{m=1}^{M}$, one for each color. All selectors $\{\mathcal{E}_{s,m}\}_{m=1}^{M}$ operate over action set C: that is, each such selector can have its arm "pulled" to select an item i to place in a slot. Though every slot s maintains M different selectors, one for each color, it only uses one at a time. The active colors $\{m_s\}_{s\in\mathcal{S}}$ are initially selected u.a.r. from $[M]$, and remain active continuously for a total K pull/feedback interactions, where $K \in \mathbb{N}$; at that point, m_s is refreshed, selected again u.a.r., bringing another selector into play. All in all, the algorithm proceeds as follows during round t.

(1) When a request $(i,p) \in \mathcal{R}^t$ is generated, it is propagated over the path p until a hit occurs; a response is then backpropagated over the path p, carrying i, and incurring a routing cost.
(2) At the same time, an additional control message is generated and propagated upstream over the entire path p. Moving upstream, it collects information from slots it traverses.
(3) After reaching designated server at the end of the path, the control message is backpropagated over the path p in the reverse direction. Every time it traverses a node $v \in p$, storage slot $s \in \mathcal{S}_v$ fetches information stored in the control message and computes a reward vector $\boldsymbol{\ell}^t(s,m_s)$. This is then fed to the active hedge selector via $\mathcal{E}_{s,m_s}$.feedback($\boldsymbol{\ell}^t(s,m_s)$).
(4) At the end of the round, we check if the arm of $\mathcal{E}_{s,m_s}$ has been pulled for a total of K times under active color m_s; if so, a new color m_s is selected u.a.r. from $[M]$.
(5) Irrespective of whether m_s changes or not, at the end of the round, each slot updates its contents via the current active hedge selector $\mathcal{E}_{s,m_s}$, by calling operation $\mathcal{E}_{s,m_s}$.arm() to choose a new item i_s to place in s.

We define the control messages exchanged, the information they carry, and the reward vectors fed to hedge selectors in Section 4.3. Only slots in a request's path need to exchange messages, provide feedback to their selectors, and (possibly) update their contents at the end of a round. Moreover, messages exchanged are of size $O(|p|)$. We allow $K \geq T$; in this case, colors are selected u.a.r. only once, at the beginning of the execution of the algorithm, and remain constant across all rounds.[4] Finally, note that updating cache contents at the end of a round does not affect the incurred cost; we remove this assumption in Section 5.

Our first main result is that DistributedTGOnline has a $(1-1/e)$-regret that grows as $O(\sqrt{T})$:

Theorem 4.1. *Consider the sequence of allocations $\{A^t\}_{t=1}^{T}$ produced by the DistributedTGOnline algorithm using hedge selectors determined by Alg. 1, with $\epsilon = \frac{1}{\bar{L}}\sqrt{\frac{\log|C|}{T}}$. Then, for all $T \geq \log|C|$ and all $K \geq 1$:*

$$\mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t)\right] \geq \beta(|\mathcal{S}|,M)\cdot \max_{A\in\mathcal{D}}\left\{\sum_{t=1}^{T} f^t(A)\right\} - 2\bar{R}\bar{L}|\mathcal{S}|M\sqrt{T\log|C|}, \tag{11}$$

where $\beta(|\mathcal{S}|,M) = 1-(1-\frac{1}{M})^M - \binom{|\mathcal{S}|}{2}M^{-1}$.

[4] In such a case, selectors corresponding to inactive colors need not be maintained, and can be discarded.

The main intuition behind the proof is to view DistributedTGOnline as a version of the offline TabularGreedy that, instead of greedily selecting a single item $i_{s,m} \in C$ per step, *it greedily selects an entire item vector* $\vec{i}_{s,m} \in C^T$ *across all rounds*, where T is the number of rounds. To cast the proof in this context, we define new objective functions f and F whose domain is over decisions across T rounds, as opposed to the original per time-slot functions (whose domain is only over one round). Due to the properties of the no-regret hedge selector, and the formal guarantees of the offline case (c.f. Thm. A.1), these new objectives attain $1 - \frac{1}{e}$ bound shown in Eq. (26), yielding the bound on the regret. The detailed proof of this theorem is provided in Appendix C. Note that for M large enough (at least $\Omega(|\mathcal{S}|^2)$), quantity $\beta(|\mathcal{S}|, M)$ can be made arbitrarily close to $1 - 1/e$. Hence, Theorem 4.1 has the following immediate corollary:

Corollary 4.2. *For any* $\delta > 0$, *there exists an* $M = \Theta(\frac{|\mathcal{S}|^2}{\delta})$ *such that the expected* $(1 - \frac{1}{e} - \delta)$*-regret of* DistributedTGOnline *is* $R_T \leq \frac{2\bar{R}\bar{L}|\mathcal{S}|^3}{\delta}\sqrt{T \log |C|}$.

DistributedTGOnline is a distributed implementation of the (centralized) online tabular greedy algorithm of Streeter et al. [77], which is itself an online implementation of the so-called tabular greedy algorithm [77], which we present in Appendix A. We depart however from [77] in several ways. First, the analysis by Streeter et al. requires that feedback is provided at every slot $s \in \mathcal{S}$. We amend this assumption, as only nodes along a path need to update their selectors/allocations in our setting. Second, we show that feedback provided to arms can be computed in a distributed fashion, using only messages along the path p, as described below in Section 4.3. These two facts together ensure that DistributedTGOnline is indeed distributed. Finally, the analysis of Streeter et al. assumes that colors are shuffled at every round, i.e., applies only to $K = 1$. We extend this to arbitrary $K \geq 1$. As we discuss in Section 5, this is instrumental to bounding regret when accounting for update costs; the later becomes $\Theta(T)$ when $K = 1$.

4.3 Control Messages, Information Exchanged, and Rewards

The algorithm is summarized in Alg. 2. We describe here the control messages, information exchanged, and reward vectors fed to hedge selectors during the generation of a request $r = (i, p) \in \mathcal{R}^t$. Let $\mathcal{S}_p = \bigcup_{v \in p} \mathcal{S}_v$ be the set of all slots in nodes in p. Let also $\mathcal{S}_{i,p} = \{s \in \mathcal{S}_p : (s, i) \in A^t\} \subseteq \mathcal{S}_p$ be the slots in path p that store the requested item $i \in C$. The control message propagated upstream towards the designated server collects both the (a) color of every slot in $\mathcal{S}_{i,p}$ and (b) the upstream cost at each node $v \in p$. Formally, the information collected by the upstream control message is

$$\mathcal{I} = \{(s, m_s)\}_{s \in \mathcal{S}_{i,p}}, \quad \text{and} \quad \mathcal{W} = \{w_v^p\}_{v \in p}, \tag{12}$$

where w_v^p is the cumulative cost of edges upstream of v on path p, i.e.,

$$w_v^p = \textstyle\sum_{k=k_p(v)}^{|p|-1} w_{p_{k+1}p_k}, \tag{13}$$

where $k_p(v) = \{1, 2, ..., |p|\}$ is position of v in p, i.e., $k_p(v) = k$ if $p_k = v$. Note that both $|\mathcal{I}|$ and $|\mathcal{W}|$ are $O(|p|)$.

Upon reaching the end of the path, a message carrying this collected information $(\mathcal{I}, \mathcal{W})$ is sent downstream to every node in the path. For each slot $s \in \mathcal{S}_p$, let:

$$\mathcal{S}_{\preceq s} = \{s' \in \mathcal{S}_{i,p} : m_{s'} < m_s \text{ or } m_{s'} = m_s, s' \prec s\} \tag{14}$$

be the slots in the path p that store i and either (a) are colored with a color smaller than m_s, or (b) are colored with m_s, and precede s in the ordering given by Eq. (1). Note that $\mathcal{S}_{\preceq s}$ can be computed having access to $\mathcal{I}$. Then, the reward vector $\boldsymbol{\ell}^r(s, m_s) \in \mathbb{R}_+^{|C|}$ fed to hedge selector $\mathcal{E}_{s,m_s}$ at slot

$s \in \mathcal{S}_p$ comprises the following coordinates:

$$\ell^r_{i'}(s, m_s) = \begin{cases} \max_{(v',j')\in\mathcal{S}_{\leq s}+s} w^p_{v'}, & \text{if } i' = i \\ \max_{(v',j')\in\mathcal{S}_{\leq s}} w^p_{v'}, & o.w. \end{cases} \quad \text{for all } i' \in C, \tag{15}$$

which captures the marginal gain of adding an element to the allocation at time t, assuming that the latter is constructed adding one slot, item, and color selection at a time (following an ordering w.r.t. colors first and slots second). This is stated formally in Lemma C.1 in Appendix C. Note again that all these quantities can be computed at every $s \in \mathcal{S}_p$ having access to $\mathcal{I}$ and $\mathcal{W}$.[5]

4.4 A Negative Result

We conclude this section with a negative result, further highlighting the importance of Theorem 4.1. DistributedTGOnline comes with adversarial guarantees; our experiments in Section 7 indicate that also works well in practice. Nevertheless, for several topologies we explore, we observe that a heuristic proposed by Ioannidis and Yeh [41], termed GreedyPathReplication, performs as well or slightly better than DistributedTGOnline in terms of time-average caching gain. As we observed this in both stationary as well as non-stationary/transient request arrivals, this motivated us to investigate further whether this algorithm also comes with any adversarial guarantees.

Our conclusion is that, despite the good empirical performance in certain settings, GreedyPathReplication does not enjoy such guarantees. In fact, the following negative result holds:

Lemma 4.3. *Consider the online policy GreedyPathReplication due to Ioannidis and Yeh [41], which is parametererized by $\beta > 0$. Then, for all $\alpha \in (0, 1]$ and all $\beta > 0$, GreedyPathReplication has $\Omega(T)$ α-regret.*

We prove this lemma in Appendix E. The adversarial counterexample we construct in the proof informed our design of experiments for which GreedyPathReplication (denoted by GRD in Section 7) performs poorly, *attaining a zero caching gain throughout the entire algorithm's execution* (see Fig. 3-5). The same examples proved hard for several other greedy/myopic policies, even though DistributedTGOnline performed close to the offline solution in these settings. Both Lemma 4.3, as well as the experimental results we present in Section 7, indicate the importance of Theorem 4.1 and, in particular, obtaining a universal bound on the regret against any adversarially selected sequence of requests.

4.5 Color Palette

To provide further intuition behind the "color palette" induced by M and the role it plays in our algorithm, we describe here in more detail how it relates to the ContinuousGreedy algorithm, the standard algorithm for maximizing submodular functions subject to a matroid constraint [15].

Intuitively, given a submodular function $f : \Omega \to \mathbb{R}_+$ and a matroid constraint set $\mathcal{D}$, the ContinuousGreedy algorithm maximizes the *multilinear relaxation* of objective, given by

$$\tilde{f}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{x}}[f(A)] = \sum_{A\subseteq\Omega} f(A) \prod_{i\in A} x_i \prod_{i\notin A} (1 - x_i). \tag{16}$$

That is, the multilinear extension $\tilde{f} : [0, 1]^{|\Omega|} \to \mathbb{R}_+$ is the expected value of the objective assuming that each element in A is sampled independently, with probability given by $x_i \in [0, 1]$, $i \in \Omega$. ContinuousGreedy first obtains a fractional solution in the matroid polytope of $\mathcal{D}$. This is constructed by incrementally growing the probability distribution parameters $\boldsymbol{x}$, starting from

[5] In practice, trading communication for space, the full $\mathcal{W}$ can also be just computed once, at the first time request (i, p) is generated, Each v can store their own w^p_v for paths that traverse them, and only weights of nodes storing i need to be included in $\mathcal{W}$ in subsequent requests.

$\boldsymbol{x} = \mathbf{0}$, with a variant of the so-called Frank-Wolfe algorithm. Then, ContinuousGreedy rounds the resulting fractional solution via, e.g., pipage rounding [2] or swap rounding [19], mapping it to set solutions. We refer the reader to Calinescu et al. [15] for a more detailed description of the algorithm.

In comparison, the color palette also creates a distribution across item selections, as implied by the M colors. When the number of colors M approaches infinity, the selection process of DistributedTGOnline also "grows" this probability distribution (starting, again, from an empty support) infinitesimally, by an increment inversely proportional to M (see also the offline version TabularGreedy in Appendix A). As M goes to infinity, this recovers the $1 - 1/e$ approximation guarantee of continuous greedy [64, 77]. For any finite M, the guarantee provided by Theorem 4.1 lies in between this guarantee and $\frac{1}{2}$ approximation of the locally greedy algorithm ($M = 1$).

5 UPDATE COSTS

We now turn our attention to incorporating update costs in our analysis. We denote by $w'_{s,i}$ the cost of fetching item i at storage slot s at the end of a round. Then, the total update cost of changing an allocation A^t to A^{t+1} at the end of round t is given by:

$$\mathrm{UC}(A^t, A^{t+1}) = \sum_{(s,i) \in A^{t+1} \setminus A^t} w'_{s,i}. \tag{17}$$

When such update costs exist, we need to account for them when adapting cache allocations. We thus incorporate them in the α-regret as follows:

$$\tilde{R}_T = \alpha \sum_{t=1}^{T} f^t(A^*) - \mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t) - \sum_{t=1}^{T-1} \mathrm{UC}(A^t, A^{t+1})\right] = R_T + \mathbb{E}\left[\sum_{t=1}^{T-1} \mathrm{UC}(A^t, A^{t+1})\right]. \tag{18}$$

Note that, in this formulation, the optimal offline policy A^* has no update cost, as it is static. We refer to $\tilde{R}_T$ as the *extended α-regret* of an algorithm. This extension corresponds to adding the expected update cost incurred by a policy $\mathcal{A}$ to its α-regret.

Unfortunately, the update costs of DistributedTGOnline (and, hence, its extended regret) grow as $\Theta(T)$ in expectation. In particular, the following lemma, proved in Appendix D, holds:

Lemma 5.1. *When DistributedTGOnline is parametrized with the hedge selector in Alg. 1, it incurs an expected update cost of $\Omega(T)$ for any choice of $\epsilon \in \mathbb{R}_+$.*

Nevertheless, the extended regret can be reduced to $O(\sqrt{T})$ by appropriately modifying Alg. 1, the no-regret-hedge selector used in slots $\mathcal{E}_{s,m}$. In particular, the linear growth in the regret is due to the independent sampling of cache contents within each round. Coupling this selection with the presently selected content can significantly reduce the update costs. More specifically, instead of selecting items independently across rounds, the new item can be selected from a probability distribution that depends on the current allocation in the cache. This conditional distribution can be design in a way that the (marginal) probability of the new item is the same as in Alg. 1, thereby yielding the same expected caching gain. On the other hand, coupling can bias the selection towards items already existing in the cache. This reduces update costs, especially when marginal distributions change slowly.

The *coupled hedge selector* described in Alg. 3 accomplishes exactly this. As seen in line 9 of Alg. 3, pulling the arm of the hedge selector is dependent on (a) the previously taken action/selected item and (b) the change in the distribution implied by weights W_i, $i \in C$. We give more intuition as to how this is accomplished in Appendix D. In short, the coupled hedge selector solves a minimum-cost flow problem via an iterative algorithm. The minimum-cost flow problem models a so-called optimal transport or earth mover distance problem [67] from $\boldsymbol{p}^t$ to $\boldsymbol{p}^{t+1}$, the distributions over catalog C

Algorithm 3: Coupled Hedge Selector

```
Input: Parameter $\epsilon \in \mathbb{R}_+$, action set $C$, horizon $T \in \mathbb{N}$.
def $\mathcal{E}$.initialize():
    Set $W_i \leftarrow 1$ and $p_i^{\text{old}} \leftarrow \frac{1}{|C|}$ for all $i \in C$
    Pick $i^{\text{old}}$ u.a.r. from $C$
def $\mathcal{E}$.feedback($\boldsymbol{\ell}$):
    Set $W_i \leftarrow W_i e^{\epsilon \ell_i}$ for all $i \in C$
def $\mathcal{E}$.correlated_arm():
    Set $p_i^{\text{new}} \leftarrow \frac{W_i}{\sum_{j \in C} W_j}$ for all $i \in C$
    Pick $i^{\text{new}} \leftarrow$ coupled_movement ($\boldsymbol{p}^{\text{old}}$,$\boldsymbol{p}^{\text{new}}$,$i^{\text{old}}$) ;        // Marginal is $\boldsymbol{p}^{\text{new}}$
    $\boldsymbol{p}^{\text{old}} \leftarrow \boldsymbol{p}^{\text{new}}$
    $i^{\text{old}} \leftarrow i^{\text{new}}$
    return $i^{\text{new}}$
def coupled_movement($\boldsymbol{p}^{\text{old}}$,$\boldsymbol{p}^{\text{new}}$,$i^{\text{old}}$ ):
    Compute $I = \{i \in C : p_i^{\text{new}} - p_i^{\text{old}} > 0\}$.
    Set $m_i \leftarrow |p_i^{\text{new}} - p_i^{\text{old}}|$, for all $i \in C$
    Compute a feasible flow $[\delta_{i,j}]_{(i,j) \in C^2}$, where $\sum_{j \in I} \delta_{i,j} = m_i$ for $i \in C \setminus I$, and
      $\sum_{i \in C \setminus I} \delta_{i,j} = m_j$ for $i \in I$, to transport $\sum_{i \in C \setminus I} m_i$ mass from the components in $I$ to the
      components in $C \setminus I$.
    $\boldsymbol{p} \leftarrow \boldsymbol{p}^{\text{old}}$.
    $i^{\text{temp}} \leftarrow i^{\text{old}}$
    for $i \in C \setminus I$ do
        for $j \in I$ do
            $\boldsymbol{p}$, $i^{\text{temp}} \leftarrow$ elementary_$\delta$movement($\boldsymbol{p}$, $i^{\text{temp}}$, $\delta_{i,j}$, $i$, $j$)
    return $i^{\text{temp}}$
def elementary_$\delta$movement($\boldsymbol{p}$, $i^{\text{temp}}$, $\delta$, $i$, $j$):
    if $i^{\text{temp}} = i$ then
        $i^{\text{temp}}, \boldsymbol{p} \leftarrow \begin{cases} i,\ \boldsymbol{p} + \delta(\boldsymbol{e}_j - \boldsymbol{e}_i) & \text{w.p. } \frac{p_i - \delta}{p_i} \\ j,\ \boldsymbol{p} + \delta(\boldsymbol{e}_j - \boldsymbol{e}_i) & \text{w.p. } \frac{\delta}{p_i} \end{cases}$
    else
        $i^{\text{temp}}, \boldsymbol{p} \leftarrow i^{\text{temp}}, \boldsymbol{p} + \delta(\boldsymbol{e}_j - \boldsymbol{e}_i)$
    return $\boldsymbol{p}$, $i^{\text{temp}}$
```

at rounds t and $t+1$, respectively, The resulting solution comprises conditional distributions for "jumps" among elements in C, which are used to determine the next item selection using the current choice: by being solutions of the minimum-cost flow problem, they incur small update cost, while ensuring that the posterior distribution after the "jump" is indeed $\boldsymbol{p}^{t+1}$.

Our second main result establishes that using this hedge selector instead of Alg. 1 in DistributedTGOnline ensures that the extended regret grows as $O(\sqrt{T})$.

Theorem 5.2. *Consider DistributedTGOnline with hedge selectors $\mathcal{E}_{s,m}$ implemented via Alg. 3 parameterized with $\epsilon = \frac{1}{L}\sqrt{\frac{\log |C|}{T}}$. Assume also that colors are updated every $K = \Omega(\sqrt{T})$ rounds.*

Table 2. Graph Topologies and Experiment Parameters. We indicate the number of nodes in each graph ($|V|$), the number of edges ($|E|$), the number of query nodes ($|Q|$), and the ranges of cache capacities c_v and edge weights w. In the last four columns we also report $f_{stationary}$, $f_{sliding}$, f_{SN}, and f_{CDN}: which are the caching gain attained by OFL with *Stationary Request*, *Sliding Popularity*, *Shot Noise*, and *CDN* trace, respectively.

topologies	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert Q\rvert$	c_v	w	$f_{stationary}$	$f_{sliding}$	f_{SN}	f_{CDN}
ER	100	1042	5	1-5	1-100	1569.93	1123.76	72.03	976.65
BT	341	680	5	1-5	1-100	5547.89	3211.63	220.58	3504.58
HC	128	896	5	1-5	1-100	2453.66	2045.69	152.38	1895.41
dtelekom	68	546	5	1-5	1-100	969.55	772.53	49.55	1005.24
GEANT	22	66	5	1-5	1-100	1564.02	981.60	66.50	1185.56
abilene	9	26	2	0-5	100	81.21	81.21	41.39	-
path	4	6	1	0-5	100	20.00	20.00	10.00	-

Then, the standard regret R_T is again bounded as in Corollary 4.2. Moreover, the update cost of DistributedTGOnline *is such that*

$$\mathbb{E}\left[\sum_{t=1}^{T-1} \mathrm{UC}(A^t, A^{t+1})\right] = O(\sqrt{T}), \tag{19}$$

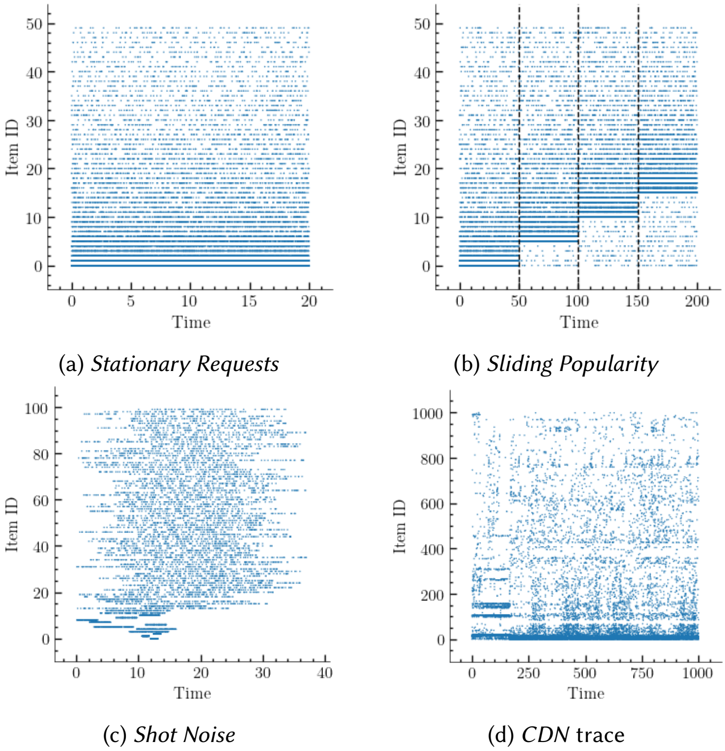
and, as a consequence, the extended regret is $\tilde{R}_T = O(\sqrt{T})$.

The proof can be found in Appendix F. We note that the theorem requires that $K = \Omega(\sqrt{T})$, i.e., that colors are shuffled infrequently. Whenever a color changes, DistributedTGOnline, the corresponding change in the active hedge selector can lead to sampling vastly different allocations than the current ones. Consequently, such changes can give rise to large update costs whenever a color is changed. The requirement that $K = \Omega(\sqrt{T})$ allows for some frequency in changes, but not, e.g., at a constant number of rounds (as, e.g., in Streeter et al. [76], where $K = 1$). Put differently, Theorem 5.2 allows for experiencing momentarily large update costs, as long as we do not surpass a budget of $O(\sqrt{T})$ color updates overall. We note that the couple hedge selector in Alg. 3 has the same time complexity as the hedge selector given in Alg. 1, which is $O(|C|)$ per iteration.

6 EXTENSIONS

Jointly Optimizing Caching and Routing. Our model can be extended to consider joint optimization of both cache and routing decisions, following an extended model by Ioannidis and Yeh [43]. Under appropriate variable transformations, the new objective is still submodular over the extended decision space. Moreover, the constraints over the routing decisions can similarly be cast as assignments to slots. Together, these allow us to directly apply DistributedTGOnline and attain the same $1 - 1/e$ approximation guarantee. We describe this extension in detail in Appendix H.

Anytime Regret Guarantee. Our algorithm assumes prior knowledge of the time horizon T; this is necessary to set parameter ϵ in Theorem 4.1. Nevertheless, we can use the well-known *doubling trick* [16] to obtain anytime regret guarantees. In short, the algorithm starts from a short horizon; at the conclusion of the horizon, the algorithm restarts, this time doubling the horizon. We show in Appendix I that, by using this doubling trick, DistributedTGOnline indeed attains an $O(\sqrt{T})$ regret, without requiring prior knowledge of T. This remains true when update costs are also considered in the regret.



(a) *Stationary Requests*

(b) *Sliding Popularity*

(c) *Shot Noise*

(d) *CDN* trace

Fig. 2. Request traces for different scenarios. Each dot indicates an access to an item in the catalog; items are ordered in an overall increasing popularity from top to bottom. In *Sliding Popularity*, popularity changes at fixed time intervals, through a cyclic shift (most popular items become least popular). In *Shot Noise*, each item remains active for a limited lifetime.

7 EXPERIMENTS

7.1 Experimental Setting

Networks. We use four synthetic graphs, namely, Erdős-Rényi (ER), balanced tree (BT), hypercube (HC), and a path (path), and three backbone network topologies: [70] Deutsche Telekom (dtelekom), GEANT, Abilene. The parameters of different topologies are shown in Tab. 2. For the first five topologies (ER–GEANT), weights w for each edge is uniformly distributed between 1–100. Each item $i \in C$ is permanently stored in a designated servers $\mathcal{D}_i$ which is designated uniformly at random (u.a.r.) from V. All nodes in V are also has c_v storage space, which is u.a.r. sampled between 1 to 5. Requests are generated from nodes Q u.a.r. selected from V. Given the source $p_1 \in Q$ and the destination $p_{|p|} \in \mathcal{D}_i$ of the request r, path p is the shortest path between them. For the remaining two topologies (abiline and path), we select parameters in a way that is described in Appendix G.

Demand. We consider three different types of synthetic request generation processes, and one trace-driven. In the *Stationary Requests* scenario (see Fig. 2(a), each $r = (i, p) \in \mathcal{R}$ is associated with an exogenous Poisson process with rate 1.0, and i is chosen from C via a power law distribution with exponent 1.2. In the *Sliding Popularity* scenario, requests are again Poisson with a different exponent 0.6, and popularities of items are periodically reshuffled (see Fig. 2(b)). In the *Shot Noise* scenario, each item is assigned a lifetime, during which it is requested according to a Poisson process; upon expiration, the item is retired (see Fig. 2(c)). In the *CDN* scenario (see Fig. 2(d)), we

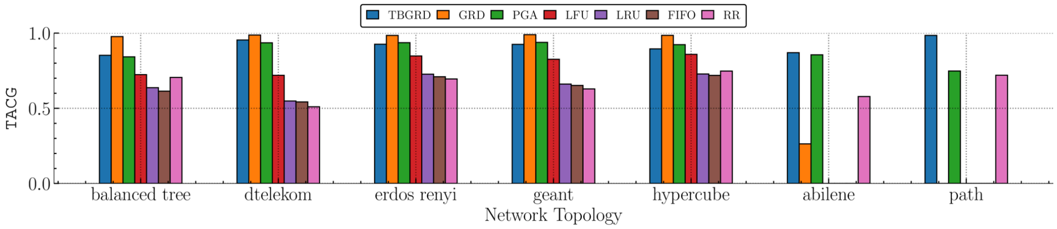


Fig. 3. TACG of different algorithms over different topologies with *Stationary Requests*. The total simulation time is 1000 time units. TBGRD, GRD, and PGA perform well in comparison to path replication policies. However, GRD and other myopic strategies attain zero TACG over abilene and path, the round-robin scenarios. In comparison, TBGRD and PGA still perform well.

generate requests using a real-life trace from a CDN provider. The trace spans 1 week, and we extract from it about 10×10^5 requests for the $N = 10^3$ most popular files.

For abiline and path, we replace the Poisson arrivals on the three synthetic traces (*Stationary Requests*, *Sliding Popularity*, *Shot Noise*) with requests generated in a round-robin manner, as described in Appendix G.2. This is designed in an adversarial fashion, that leads to poor performance for greedy/myopic algorithms.

Algorithms. We implement the following online algorithms [6]:

- Path replication with least recently used (LRU), least frequently used (LFU), first-in-first-out (FIFO), and random-replacement (RR) eviction policies: In all these algorithms, when responses are back-propagated over the reverse path, all nodes they encounter store requested item, evicting items according to one of the aforementioned policies.
- Projected gradient ascent (PGA): This is the distributed, adaptive algorithm oringinally proposed by Ioannidis and Yeh [41]. This is attains an $(1 - 1/e)$-approximation guarantee in expectation when requests are stationary, but comes with no guarantee against adversarial requests. Similar to our setting, it also operated in rounds, at the end of which contents are shuffled.
- Greedy path replication (GRD): This is a heuristic, also proposed by Ioannidis and Yeh [41]. Though it performs well in many cases, we prove in Appendix D that its $(1 - 1/e)$-regret is $\Omega(T)$ in the worst case.
- DistributedTGOnline (TBGRD): this is our proposed algorithm. We implement it with both independent hedge selector shown in Algorithm 1 and coupled hedge selector in Algorithm 3.

Unless indicated otherwise, we set $\epsilon = 0.005$, number of colors $M = 100$, $\bar{R} = 1$, and $K = T$ for TBGRD. For PGA and GRD, we explore parameters γ and β range from 0.005-5 and 0.005-1 individually, and pick the optimal values. In experiments where we do not measure update costs, we implement TBGRD with the independent hedge selector (Alg. 1), as it yields the same performance as the coupled hedge selector (Alg. 3) in expectation (see also Fig. 9(a) and 9(b)).

Finally, we also implement the offline algorithm (OFL) by Ioannidis and Yeh [41], and use the resulting $(1 - 1/e)$-approximate solution as baseline (see metrics below).

Performance Metrics. We use normalized time-average cache gain (TACG) as the metric to measure the performance of different algorithms. More specifically, leveraging PASTA [78], we measure $f^{t_s}(A^{t_s})$ at epochs t_s generated by a Poisson process with rate 1.0 for 5000 time slots, and average these measurements. To compare performance across different topologies, we normalize

[6] Our code is publicly available at https://github.com/neu-spiral/OnlineCache.

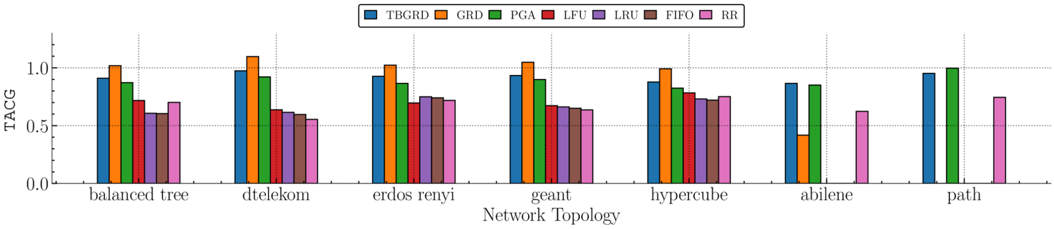


Fig. 4. TACG of different algorithms over different topologies with *Sliding Popularity*. The total simulation time is 1000 time units. TBGRD, GRD, and PGA again outperform path replication algorithms; GRD sometimes even outperforms the (static) OFL solution, attaining a normalized TACG larger than one. However, GRD and several path replication algorithms again fail catastrophically over the abilene and path scenarios, while TBGRD and PGA again attain a normalized TACG close to one.

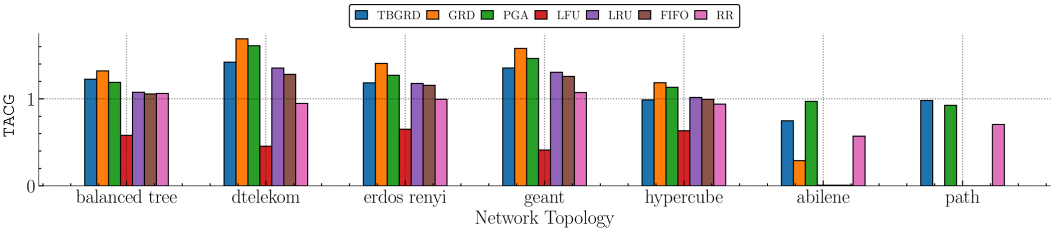


Fig. 5. TACG of different algorithms over different topologies with *Shot Noise*. We again observe TBGRD, GRD, and PGA perform well in this non-stationary request arrival setting. Moreover, several algorithms outperform the (static) offline solution OFL in this setting. Again, GRD and other myopic path replication policies fail over abilene and path, while TBGRD and PGA still attain a non-zero TACG.

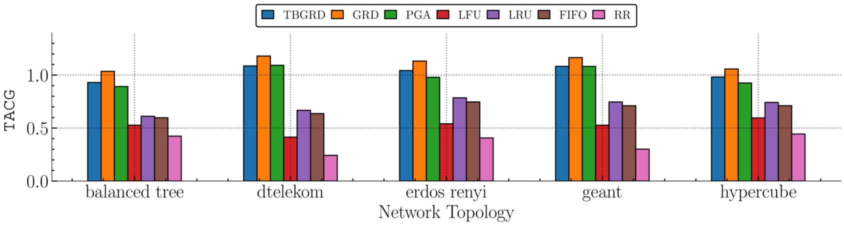


Fig. 6. TACG of different algorithms over different topologies with *CDN* trace. The total simulation time is 2000 time units. We again observe that TBGRD, GRD, and PGA outperform path replication policies.

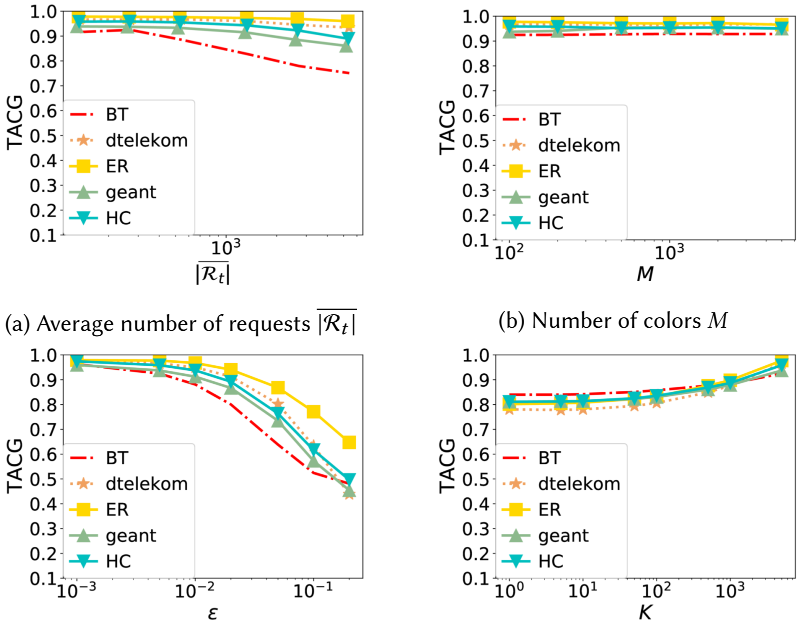
the average by $f_{\texttt{OFL}}$, the caching gain attained by OFL, yielding:

$$\text{TACG} = \tfrac{1}{T_s f_{\texttt{OFL}}} \sum_{t_s=1}^{T_s} f^{t_s}(A^{t_s}). \tag{20}$$

The corresponding $f_{\texttt{OFL}}$ values are reported in Table 2. We also measure the cumulative update cost (CUC) of TBGRD over time under the hedge and coupled hedge selectors, i.e.,

$$\text{CUC} = \sum_{t=1}^{T-1} \text{UC}(A^t, A^{t+1}), \tag{21}$$

where we measure the instantaneous update cost UC using (17) with weights set to 1.



(a) Average number of requests $\overline{|\mathcal{R}_t|}$

(b) Number of colors M

(c) Parameter ϵ

(d) Color update period K

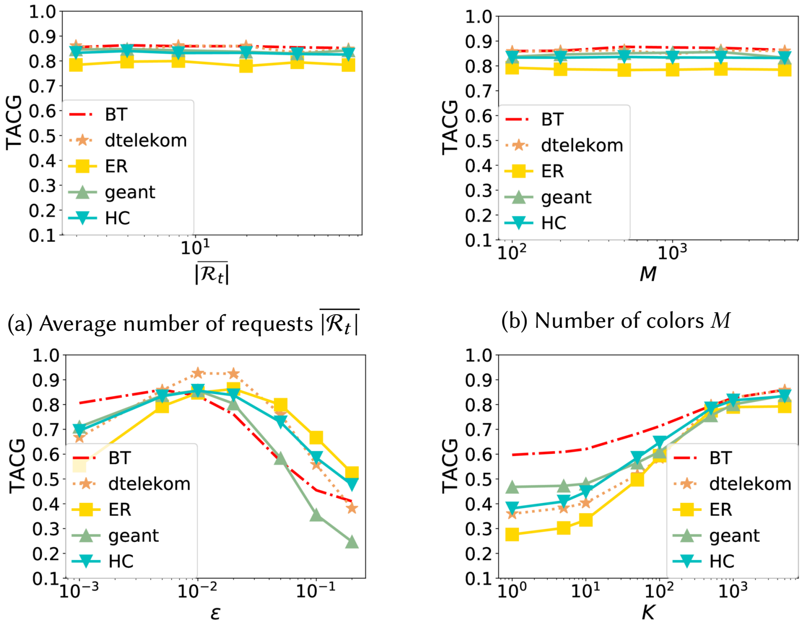
Fig. 7. TACG vs. different parameters under *Stationary Request*. As the average number of requests increases, TACG decreases. Number of colors does not affect a lot. As ϵ increases, TACG decreases. As color update period increases, TACG increases.

7.2 Results

TACG Comparison. Figures 3-5 show the performance of different algorithms w.r.t. TACG across multiple topologies, for different synthetic traces (*Stationary*, *Sliding Popularity*, and *Shot Noise*, respectively). For GRD here, we explore parameters ϵ range from 0.0001-1, and pick the optimal values. We observe that TBGRD, GRD, and PGA have similar performance across topologies on all three traces for the first five topologies, with GRD being slightly higher performing than the other two; nevertheless, on the last two topologies, that have been designed to lead to poor performance for myopic/greedy strategies, both GRD and other myopic strategies (e.g., LFU, LRU, and FIFO) are stymied, attaining a zero caching gain throughout. This also verifies the suboptimality of GRD stated in Lemma 4.3. In contrast, TBGRD and PGA still attain a TACG close to the offline value; not surprisingly RR also has a suboptimal, but non-zero gain in these scenarios as well.

The more a trace departs from stationarity, the more the performance of OFL degrades: As seen in Tab. 2, the caching gain obtained by OFL consistently across the different topologies has the highest value in the *Stationary* trace, then decreases as we change to *CDN*, *Sliding Popularity SN* traces, in that order.

We also note that in the *Shot Noise* case several algorithms attain a normalized TACG that is higher than 1. This indicates that the dynamic algorithms beat the static offline policy in this setting. The above observations largely carry over to the *CDN* trace, shown in Figure 6, for which however we do not consider the two round-robin demand scenarios (abilene and path), as the demand is driven by the trace.



(a) Average number of requests $\overline{|\mathcal{R}_t|}$

(b) Number of colors M

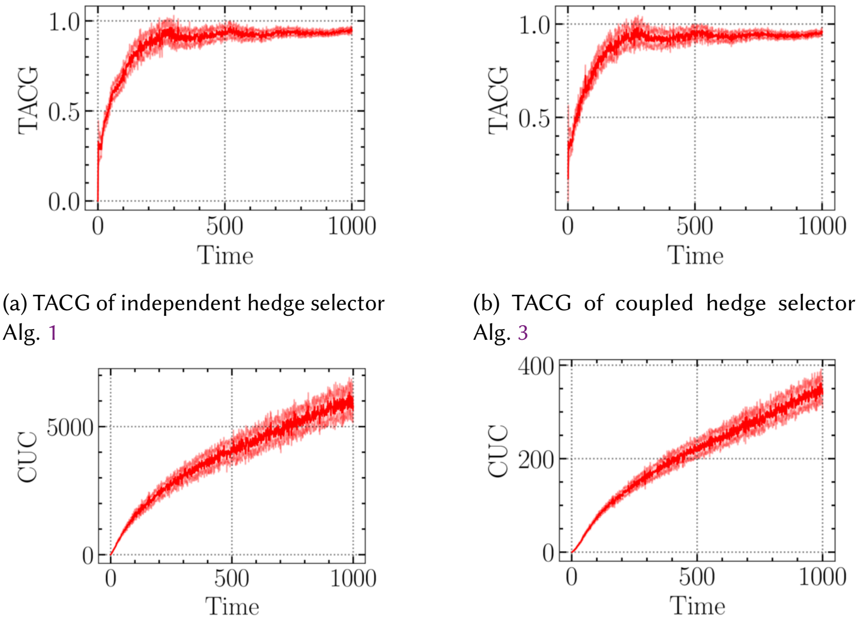
(c) Parameter ϵ

(d) Color update period K

Fig. 8. TACG v.s. different parameter with *Sliding Popularity* scenario. The average number of requests and number of colors do not affect performance significantly. The optimal ϵ is at about 0.01 for multiple topologies; this selection corresponds to a decay rate of the item selection probability that is most appropriate for the popularity refreshing period of these traces. As the color update period increases, TACG increases.

Impact of Different Parameters. We explore the effect of different parameters in TBGRD with both stationary and sliding popularity requests in Figures 7 and 8, respectively, for five different topologies. We plot the normalized TACG with different values of colors M, parameter ϵ, color update period K, and average number of requests per round $\overline{|\mathcal{R}_t|}$. For the latter, we select a round duration of B time units, and group all requests within a duration together in to a single request set $\mathcal{R}_t$; note that, due to stochasticity, the number of requests varies at each round t. In general, the normalized TACG for stationary requests is slightly higher than for sliding popularity. This is expected, as stationary requests are easier to learn. The number of colors does not affect the performance of algorithm a lot, shown in Fig. 7(b) and 8(b). From both Fig. 7(a), we see that smaller request set size leads to better TACG, which again makes sense: that more frequent cache updates are, the faster they adapt to current requests. Besides this, we see that $\overline{|\mathcal{R}_t|}$ has bigger impact under stationary requests, while the sliding window scenario is less affected by varying this parameter. We also observe in Fig. 7(c) that greater ϵ values lead to worse performance in the stationary setting; however in the sliding popularity setting, shown in 8(c), the optimal ϵ is at about 0.01 for multiple topologies; this selection corresponds to a decay rate of the item selection probability that is most appropriate for the popularity refreshing period of these traces. Finally, even though higher K is better on both Fig. 7(d) and 8(d), we see more variability/bigger impact of this selection in the sliding popularity trace.

Update Costs. Recall from Theorem 5.2 both the (independent) hedge selector and the coupled hedge selector lead to same caching gain in expectation. This also verified experimentally by results



(a) TACG of independent hedge selector Alg. 1

(b) TACG of coupled hedge selector Alg. 3

(c) Cumulative update cost (CUC) of independent hedge selector Alg. 1

(d) Cumulative update cost (CUC) of coupled hedge selector Alg. 3

Fig. 9. TACG and CUC of DistributedTGOnline over *Sliding Popularity* trace/dtelekom. The learning rate is $\epsilon = 5 \times 10^4$. Values reported are averaged over 30 experiments with different random seeds.

of Fig. 9 (a) and (b): we observe that both hedge selectors lead to almost identical TACG on the sliding popularity trace. We also observe that the cumulative update cost (CUC), shown 9(c) and Fig. 9(d), is vastly different across the two selectors: within the duration of the simulation, the CUC of the hedge selector is more than 15× the CUC of the coupled hedge selector.

8 CONCLUSION

We propose a distributed, online algorithm that achieves sublinear $(1-1/e)$-regret for the adversarial caching gain maximization problem, even when accounting for update costs. An interesting future research direction is to provide regret guarantees for the class of path replication algorithms. These algorithms are appealing precisely because they do not involve updates that happen separately from the normal response traffic: whenever a response packet carrying an item traverses a cache, the latter makes a decision of whether to cache this content or not on the spot. This restricts the type of allocations that an online algorithm can construct at any point in time, but does not incur any additional cost beyond the one generated by response traffic. This property makes path replication algorithms quite popular in practice [24, 44, 52]. Our proof that GreedyPathReplication has linear regret (see Lemma 4.3) is a negative result in this direction. Nevertheless, determining whether a path replication algorithm that has sublinear α-regret exists remains an interesting open problem, from both a theoretical and practical point of view. Another important future research direction is to consider dynamic regret [81], whereby the performance of a policy is compared to a dynamic optimum. Dynamic regret was studied under different settings of online convex optimization [8, 36, 37, 81], multi-armed bandits [9, 48, 49, 58], and non-stationary reinforcement learning [28, 45, 60], and would be interesting to apply to our setting.

9 ACKNOWLEDGMENTS

The authors gratefully acknowledge support from the National Science Foundation (grants 1718355, 2107062, and 2112471), as well as from Inria under the exploratory action MAMMALS.

REFERENCES

[1] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. 2014. Taming the monster: A fast and simple algorithm for contextual bandits. In *International Conference on Machine Learning*. PMLR, 1638–1646.

[2] Alexander A Ageev and Maxim I Sviridenko. 2004. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization* 8, 3 (2004), 307–328. https://doi.org/10.1023/b:joco.0000038913.96607.c2

[3] Sara Alouf, Nicaise Choungmo Fofack, and Nedko Nedkov. 2016. Performance models for hierarchy of caches: Application to modern DNS caches. *Performance Evaluation* 97 (2016), 57–82.

[4] Lachlan Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. 2013. A Tale of Two Metrics: Simultaneous Bounds on Competitiveness and Regret. *SIGMETRICS Perform. Eval. Rev.* 41, 1 (June 2013), 329–330. https://doi.org/10.1145/2494232.2465533

[5] Sanjeev Arora, Elad Hazan, and Satyen Kale. 2012. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing* 8, 1 (2012), 121–164.

[6] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. 2009. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science* 410, 19 (2009), 1876–1902.

[7] Daniel S Berger, Philipp Gland, Sahil Singla, and Florin Ciucu. 2014. Exact analysis of TTL cache networks. *Performance Evaluation* 79 (2014), 2–23. https://doi.org/10.1016/j.peva.2014.07.001

[8] Omar Besbes, Yonatan Gur, and Assaf Zeevi. 2015. Non-stationary stochastic optimization. *Operations research* 63, 5 (2015), 1227–1244.

[9] Lilian Besson and Emilie Kaufmann. 2019. The generalized likelihood ratio test meets klucb: an improved algorithm for piece-wise non-stationary bandits. *Proceedings of Machine Learning Research vol XX* 1 (2019), 35.

[10] Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. 2020. Fundamental Limits on the Regret of Online Network-Caching. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 2, Article 25 (June 2020), 31 pages. https://doi.org/10.1145/3392143

[11] Allan Borodin, Nathan Linial, and Michael E. Saks. 1992. An Optimal On-line Algorithm for Metrical Task System. *J. ACM* 39, 4 (Oct. 1992), 745–763. https://doi.org/10.1145/146585.146588

[12] Sem Borst, Varun Gupta, and Anwar Walid. 2010. Distributed caching algorithms for content distribution networks. In *IEEE Conference on Computer Communications (INFOCOM 2010)*. 1–9. https://doi.org/10.1109/INFCOM.2010.5461964

[13] Sébastien Bubeck. 2015. Convex Optimization: Algorithms and Complexity. *Foundations and Trends in Machine Learning* 8, 3-4 (2015), 231–357.

[14] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. 2007. Maximizing a submodular set function subject to a matroid constraint. In *International Conference on Integer Programming and Combinatorial Optimization*. Springer, 182–196.

[15] Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. 2011. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.* 40, 6 (2011), 1740–1766.

[16] Nicolo Cesa-Bianchi and Gabor Lugosi. 2006. *Prediction, Learning, and Games.* Cambridge University Press. https://doi.org/10.1017/CBO9780511546921

[17] TH Hubert Chan, Zhiyi Huang, Shaofeng H-C Jiang, Ning Kang, and Zhihao Gavin Tang. 2017. Online submodular maximization with free disposal: Randomization beats for partition matroids. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1204–1223.

[18] Hao Che, Ye Tung, and Zhijun Wang. 2002. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications* 20, 7 (2002), 1305–1314. https://doi.org/10.1109/JSAC.2002.801752

[19] Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. 2010. Dependent randomized rounding via exchange properties of combinatorial structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, 575–584.

[20] Lin Chen, Christopher Harshaw, Hamed Hassani, and Amin Karbasi. 2018. Projection-Free Online Optimization with Stochastic Gradient: From Convexity to Submodularity. In *International Conference on Machine Learning*. 814–823.

[21] Lin Chen, Hamed Hassani, and Amin Karbasi. 2018. Online Continuous Submodular Maximization. In *International Conference on Artificial Intelligence and Statistics*. 1896–1905.

[22] Weibo Chu, Mostafa Dehghan, John CS Lui, Don Towsley, and ZhiLi Zhang. 2018. Joint cache resource allocation and request routing for in-network caching services. *Computer Networks* 131 (2018), 1–14. https://doi.org/10.1016/j.comnet.2017.11.009

[23] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 208–214.

[24] Edith Cohen and Scott Shenker. 2002. Replication strategies in unstructured peer-to-peer networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 32. ACM, 177–190.

[25] Mostafa Dehghan, Laurent Massoulie, Don Towsley, Daniel Menasche, and YC Tay. 2016. A utility optimization approach to network cache design. In *IEEE Conference on Computer Communications (INFOCOM 2016)*.

[26] Miroslav Dudik, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang. 2011. Efficient optimal learning for contextual bandits. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. 169–178.

[27] Ronald Fagin. 1977. Asymptotic miss ratios over independent references. *J. Comput. System Sci.* 14, 2 (1977), 222 – 250.

[28] Yingjie Fei, Zhuoran Yang, Zhaoran Wang, and Qiaomin Xie. 2020. Dynamic Regret of Policy Optimization in Non-Stationary Environments. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[29] Yuval Filmus and Justin Ward. 2014. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM J. Comput.* 43, 2 (2014), 514–542.

[30] Marshall L Fisher, George L Nemhauser, and Laurence A Wolsey. 1978. An analysis of approximations for maximizing submodular set functions—II. In *Polyhedral combinatorics*. Springer, 73–87.

[31] Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. 2014. Performance evaluation of hierarchical TTL-based cache networks. *Computer Networks* 65 (2014), 212 – 231. https://doi.org/10.1016/j.comnet.2014.03.006

[32] Christine Fricker, Philippe Robert, and James Roberts. 2012. A versatile and accurate approximation for LRU cache performance. In *2012 24th International Teletraffic Congress (ITC 24)*. IEEE, 1–8.

[33] Daniel Golovin, Andreas Krause, and Matthew Streeter. 2014. Online submodular maximization under a matroid constraint with application to learning assignments. *arXiv preprint arXiv:1407.1082* (2014).

[34] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. 2010. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *International Workshop on Internet and Network Economics*. Springer, 246–257.

[35] Salah Eddine Hajri and Mohamad Assaad. 2017. Energy efficiency in cache-enabled small cell networks with adaptive user clustering. *IEEE Transactions on Wireless Communications* 17, 2 (2017), 955–968.

[36] Eric Hall and Rebecca Willett. 2013. Dynamical models and tracking regret in online convex programming. In *International Conference on Machine Learning*. PMLR, 579–587.

[37] Eric C Hall and Rebecca M Willett. 2015. Online convex optimization in dynamic environments. *IEEE Journal of Selected Topics in Signal Processing* 9, 4 (2015), 647–662.

[38] Hamed Hassani, Mahdi Soltanolkotabi, and Amin Karbasi. 2017. Gradient methods for submodular maximization. In *Advances in Neural Information Processing Systems*. 5841–5851.

[39] Elad Hazan et al. 2016. Introduction to online convex optimization. *Foundations and Trends® in Optimization* 2, 3-4 (2016), 157–325.

[40] Stratis Ioannidis, Laurent Massoulié, and Augustin Chaintreau. 2010. Distributed caching over heterogeneous mobile networks. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. 311–322.

[41] Stratis Ioannidis and Edmund Yeh. 2016. Adaptive caching networks with optimality guarantees. *ACM SIGMETRICS Performance Evaluation Review* 44, 1 (2016), 113–124.

[42] Stratis Ioannidis and Edmund Yeh. 2018. Adaptive caching networks with optimality guarantees. *IEEE/ACM Transactions on Networking* 26, 2 (2018), 737–750. https://doi.org/10.1109/TNET.2018.2793581

[43] Stratis Ioannidis and Edmund Yeh. 2018. Jointly Optimal Routing and Caching for Arbitrary Network Topologies. *IEEE Journal on Selected Areas in Communications* 36, 6 (2018), 1258–1275. https://doi.org/10.1109/JSAC.2018.2844981

[44] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. 2009. Networking named content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. ACM, 1–12. https://doi.org/10.1145/1658939.1658941

[45] Thomas Jaksch, Ronald Ortner, and Peter Auer. 2010. Near-optimal Regret Bounds for Reinforcement Learning. *Journal of Machine Learning Research* 11, 4 (2010).

[46] Predrag R. Jelenkovic. 1999. Asymptotic Approximation of the Move-to-Front Search Cost Distribution and Least-Recently Used Caching Fault Probabilities. *The Annals of Applied Probability* 9, 2 (1999), 430–464.

[47] Bo Jiang, Philippe Nain, and Don Towsley. 2018. On the Convergence of the TTL Approximation for an LRU Cache Under independent Stationary Request Processes. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 3, 4 (2018), 1–31. https://doi.org/10.1145/3239164

[48] Zohar S Karnin and Oren Anava. 2016. Multi-armed bandits: Competing with optimal sequences. *Advances in Neural Information Processing Systems* 29 (2016), 199–207.

[49] N Bora Keskin and Assaf Zeevi. 2017. Chasing demand: Learning and earning in a changing environment. *Mathematics of Operations Research* 42, 2 (2017), 277–307.

[50] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. 2008. Multi-armed bandits in metric spaces. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 681–690.

[51] Elias Koutsoupias. 2009. The k-server Problem. *Computer Science Review* 3, 2 (May 2009), 105–118. https://doi.org/10.1016/j.cosrev.2009.04.002

[52] Nikolaos Laoutaris, Sofia Syntila, and Ioannis Stavrakakis. 2004. Meta algorithms for hierarchical Web caches. In *IEEE International Conference on Performance, Computing, and Communications, 2004*. 445–452. https://doi.org/10.1109/PCCC.2004.1395054

[53] Emilio Leonardi and Giovanni Neglia. 2018. Implicit Coordination of Caches in Small Cell Networks Under Unknown Popularity Profiles. *IEEE Journal on Selected Areas in Communications* 36, 6 (June 2018), 1276–1285. https://doi.org/10.1109/JSAC.2018.2844982

[54] Jian Li, Truong Khoa Phan, Wei Koong Chai, Daphne Tuncer, George Pavlou, David Griffin, and Miguel Rio. 2018. Dr-cache: Distributed resilient caching with latency guarantees. In *IEEE Conference on Computer Communications (INFOCOM 2018)*. 441–449. https://doi.org/10.1109/INFOCOM.2018.8486316

[55] Yuanyuan Li and Stratis Ioannidis. 2020. Universally Stable Cache Networks. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE.

[56] Boxi Liu, Konstantinos Poularakis, Leandros Tassiulas, and Tao Jiang. 2019. Joint Caching and Routing in Congestible Networks of Arbitrary Topology. *IEEE Internet of Things Journal* 6, 6 (2019), 10105–10118. https://doi.org/10.1109/JIOT.2019.2935742

[57] Yuezhou Liu, Yuanyuan Li, Qian Ma, Stratis Ioannidis, and Edmund Yeh. 2020. Fair caching networks. *Performance Evaluation* (2020). https://doi.org/10.1016/j.peva.2020.102138.

[58] Haipeng Luo, Chen-Yu Wei, Alekh Agarwal, and John Langford. 2018. Efficient contextual bandits in non-stationary worlds. In *Conference On Learning Theory*. PMLR, 1739–1776.

[59] Mark Manasse, Lyle McGeoch, and Daniel Sleator. 1988. Competitive Algorithms for On-Line Problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, New York, NY, USA, 322–333. https://doi.org/10.1145/62212.62243

[60] Weichao Mao, Kaiqing Zhang, Ruihao Zhu, David Simchi-Levi, and Tamer Basar. 2021. Near-Optimal Model-Free Reinforcement Learning in Non-Stationary Episodic MDPs. In *International Conference on Machine Learning*. PMLR, 7447–7458.

[61] Aryan Mokhtari, Hamed Hassani, and Amin Karbasi. 2018. Conditional Gradient Method for Stochastic Submodular Maximization: Closing the Gap. In *International Conference on Artificial Intelligence and Statistics*. 1886–1895.

[62] Samrat Mukhopadhyay and Abhishek Sinha. 2021. Online Caching with Optimal Switching Regret. In *2021 IEEE International Symposium on Information Theory (ISIT)*. 1546–1551. https://doi.org/10.1109/ISIT45174.2021.9517925

[63] Giovanni Neglia, Emilio Leonardi, Guilherme Iecker Ricardo, and Thrasyvoulos Spyropoulos. 2021. A Swiss Army Knife for Online Caching in Small Cell Networks. *IEEE/ACM Transactions on Networking* (2021).

[64] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming* 14, 1 (1978), 265–294.

[65] Debjit Paria, Krishnakumar, and Abhishek Sinha. 2020. Caching in Networks without Regret. arXiv:2009.08228 [cs.IT]

[66] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis. 2019. Learning to Cache With No Regrets. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 235–243.

[67] Gabriel Peyré, Marco Cuturi, et al. 2019. Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning* 11, 5-6 (2019), 355–607.

[68] Konstantinos Poularakis, George Iosifidis, Vasilis Sourlas, and Leandros Tassiulas. 2016. Exploiting caching and multicast for 5G wireless networks. *IEEE Transactions on Wireless Communications* 15, 4 (2016), 2995–3007.

[69] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. 2008. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*. 784–791.

[70] Dario Rossi and Giuseppe Rossini. 2011. Caching performance of content centric networks under multi-path routing (and more). *Relatório técnico, Telecom ParisTech* (2011), 1–6.

[71] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2021. No-Regret Caching via Online Mirror Descent. arXiv:2101.12588 [cs.LG]

[72] Shai Shalev-Shwartz. 2012. Online Learning and Online Convex Optimization. *Found. Trends Mach. Learn.* 4, 2 (Feb. 2012), 107–194. https://doi.org/10.1561/2200000018

[73] Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. 2013. Femtocaching: Wireless content delivery through distributed caching helpers. *IEEE Transactions on Information Theory*

Algorithm 4: TabularGreedy

```
Input: Integer $M$, set $C$, function $f$.
set $\tilde{A} \leftarrow \emptyset$.
for $m \leftarrow 1$ to $M$ do
    foreach $s \in \mathcal{S}$ do
        Find $i_{s,m}$ s.t. $F(\tilde{A} + (s, i_{s,m}, m)) \geq \max_{i \in C} F(\tilde{A} + (s, i, m)) - \epsilon_{s,m}$
        $\tilde{A} \leftarrow \tilde{A} + (s, i_{s,m}, m)$
foreach $s \in \mathcal{S}$ do
    independently choose $m_s$ uniformly at random from $M$
return $\texttt{sample}_{\boldsymbol{m}}(\tilde{A})$
```

59, 12 (2013), 8402–8413.

[74] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2021. No-Regret Caching via Online Mirror Descent. In *IEEE International Conference on Communications (ICC).*

[75] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM* 28, 2 (Feb. 1985), 202–208. https://doi.org/10.1145/2786.2793

[76] Matthew Streeter and Daniel Golovin. 2008. An online algorithm for maximizing submodular functions. *Advances in Neural Information Processing Systems* 21 (2008), 1577–1584.

[77] Matthew Streeter, Daniel Golovin, and Andreas Krause. 2009. Online learning of assignments. *Advances in neural information processing systems* 22 (2009), 1794–1802.

[78] Ronald W Wolff. 1982. Poisson arrivals see time averages. *Operations Research* 30, 2 (1982), 223–231.

[79] Zhengyu Yang, Danlin Jia, Stratis Ioannidis, Ningfang Mi, and Bo Sheng. 2018. Intermediate data caching optimization for multi-stage and parallel big data frameworks. In *2018 IEEE 11th International Conference on Cloud Computing*. 277–284. https://doi.org/10.1109/CLOUD.2018.00042

[80] Mingrui Zhang, Lin Chen, Hamed Hassani, and Amin Karbasi. 2019. Online Continuous Submodular Maximization: From Full-Information to Bandit Feedback.. In *NeurIPS*.

[81] Martin Zinkevich. 2003. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning* (Washington, DC, USA) *(ICML'03)*. AAAI Press, 928–935.

A TABULAR GREEDY ALGORITHM

We present here TabularGreedy [77], a polynomial time algorithm for solving Problem (7) within a $(1 - 1/e)$-approximation. This differs from the (more common) continuous greedy algorithm [15] in that it operates in the discrete rather than continuous domain, even though both algorithms involve randomization. It serves as the basis for the online algorithm by Streeter et al. [77]. A key departure from continuous greedy is the use of randomization via colors assigned to each slot, which also manifest in the online version of the algorithm.

For any set $\tilde{A} \subseteq \mathcal{S} \times C \times [M]$ and vector $\boldsymbol{m} = [m_s]_{s \in \mathcal{S}} \in [M]^{|\mathcal{S}|}$, let:

$$\texttt{sample}_{\boldsymbol{m}}(\tilde{A}) = \left\{ (s, i) \in \mathcal{S} \times C : (s, i, m_s) \in \tilde{A} \right\}. \tag{22}$$

Intuitively, the colored allocation $\tilde{A} \subseteq \mathcal{S} \times C \times [M]$ is an allocation of items to slots, additionally parameterized by colors. Given the color vector $\boldsymbol{m}$, assigning colors to slots, $\texttt{sample}_{\boldsymbol{m}}$ acts as a selector, producing an (uncolored) allocation $A \subseteq \mathcal{S} \times C$. Let $F(\tilde{A})$ be the expected value of $f(\texttt{sample}_{\boldsymbol{m}}(\tilde{A}))$ when each color m_s is selected independently and u.a.r. from $[M]$; formally,

$$F(\tilde{A}) = \mathbb{E}\left[f(\texttt{sample}_{\boldsymbol{m}}(\tilde{A})) \right] = \frac{1}{M^{|\mathcal{S}|}} \sum_{\boldsymbol{m}' \in [M]^{|\mathcal{S}|}} f(\texttt{sample}_{\boldsymbol{m}'}(\tilde{A})). \tag{23}$$

The procedure is summarized in Algorithm 4. TabularGreedy constructs a set of triplets $\tilde{A} \subseteq \mathcal{S} \times C \times [M]$ greedily; that is, starting from an empty set, it iterates over all colors and storage slots in an arbitrary order, and places items to (colored) slots by greedily maximizing the extended function F. Formally, in the iteration over color $m \in [M]$ and slot $s \in S$, the algorithm extends $\tilde{A}$ via:

$$i_{s,m} = \arg\max_{i \in C}\{F(\tilde{A} + (s, i, m))\} \tag{24a}$$

$$\tilde{A} \leftarrow \tilde{A} + (s, i_{s,m}, m), \tag{24b}$$

where, for legibility, we use $\tilde{A} + o$ to indicate $\tilde{A} \cup \{o\}$. Finally, the algorithm returns allocation $S = \texttt{sample}_{\boldsymbol{m}}(\tilde{A})$, where colors in vector $\boldsymbol{m}$ are selected u.a.r. from $[M]$.

Note that the same node would not cache the same content multiple times. Indeed, as shown in Eq. (24), the algorithm extends the set of triplets $\tilde{A}$, in the iteration over color m and slot s, by the maximizer (s, i, m) of $F(\tilde{A} + (s, i, m))$. To be more specific, in the same node, if it is possible that an item i is repeatedly cached, then one of the triplets (s, i, m) could not be the maximizer in some iteration, since a different item i' could achieve greater or equal cache gain than i. This internally avoids duplicate cache in one nodes.

The following theorem characterizes the approximation guarantee of the solution produced by TabularGreedy; the theorem allows for the case where the greedy item selection $i_{s,m}$ by (24a) is inexact, and the selected item is suboptimal by an offest $\epsilon_{s,m}$. Let $\tilde{A}^-_{s,m}$ equal $\tilde{A}$ just before $(s, i_{s,m}, m)$ is added at iteration m, s, i.e., $\tilde{A}^-_{s,m} = \{(s', i_{s',m'}, m') : s' \in \mathcal{S}, m' < m\} \cup \{(s', i_{s',m}, m) : s' \prec s\}$:

Theorem A.1. *(Theorem 13 in [33]) Suppose f is monotone submodular. Consider an arbitrary ordering of colors $m \in [M]$ and slots $s \in \mathcal{S}$, and consider the sequence of sets constructed by* TabularGreedy *when $i_{s,m} \in C$ in Eq. (24a) is such that:*

$$F(\tilde{A} + (s, i_{s,m}, m)) \geq \max_{i \in C} F(\tilde{A}^-_{s,m} + (s, i, m)) - \epsilon_{s,m}, \tag{25}$$

for some $\epsilon_{s,m} \geq 0$. Then, the final set in the sequence $\tilde{A} \subseteq \mathcal{S} \times C \times [M]$ satisfies:

$$F(\tilde{A}) \geq \beta(|\mathcal{S}|, M) \cdot \max_{A \in \mathcal{D}} f(A) - \sum_{s \in \mathcal{S}} \sum_{m=1}^{M} \epsilon_{s,m}, \tag{26}$$

where $\beta(|\mathcal{S}|, M) = 1 - (1 - \frac{1}{M})^M - \binom{|\mathcal{S}|}{2} M^{-1}$.

The importance of accounting for inexact greedy selection lies in the fact that expectation F is hard to compute exactly, and is typically approximated by sampling, i.e., via $\tilde{F}(\tilde{A}) = \frac{1}{L}\sum_{l=1}^{L} f(\texttt{sample}_{\boldsymbol{m}_l}(\tilde{S}))$, were $\boldsymbol{m}_l$ are sampled u.a.r. The theorem implies that by selecting a large enough M (in particular, larger than $\Theta(|\mathcal{S}|^2)$, and L, the approximation guarantee can get arbitrarily close to $1 - 1/e$.

B FORMAL GUARANTEES OF HEDGE SELECTOR ALG. 1

Recall that, at each time t, the hedge selector $\mathcal{E}$ defined by Alg. 1 picks an action i^t from finite set C and subsequently observes an adversarially selected vector of rewards $\boldsymbol{\ell}^t = [\ell^t_i]_{i \in C} \in \mathbb{R}_+^{|C|}$, where ℓ^t_i is the reward for choosing action $i \in C$ at round t. The selector then accrues reward $\ell^t_{i^t}$, i.e., the reward associated with the action i^t it selected previously. We note that the hedge selector operates in the full-information (rather than the classic bandit) setting: all action rewards in C are observed. The regret R_T of hedge selector $\mathcal{E}$ is:

$$R_T = \sum_{t=1}^{T} \ell^t_{i^*} - \mathbb{E}[\sum_{t=1}^{T} \ell^t_{i^t}], \tag{27}$$

where i^* is the best selection in hindsight, i.e., $i^* = \arg\max_{i \in C} \sum_{t=1}^{T} \ell_i^t$.

The following lemma is classic; we note that it follows immediately from Theorem 1.5 in Hazan [39]. We reprove it here for completeness.

Lemma B.1 ([5, 39]). *Assume that every action's reward is bounded by $\bar{L} \in \mathbb{R}_+$. Let $\epsilon = \frac{1}{\bar{L}}\sqrt{\frac{\log|C|}{T}}$. Then, for all $T \geq \log|C|$, the regret of hedge selector $\mathcal{E}$ defined by (9) and (10) is s.t.:*

$$R_T \leq 2\bar{L}\sqrt{T \log|C|}. \tag{28}$$

Proof. Observe that for $T \geq \log|C|$, we have that

$$\epsilon = \frac{1}{\bar{L}}\sqrt{\frac{\log|C|}{T}} \in [0, \frac{1}{\bar{L}}]. \tag{29}$$

Let $\Phi^t = \sum_{i \in C} W_i^t$, $\boldsymbol{p}^t = [p_i^t]_{i \in C} \in \mathbb{R}^{|C|}$.

$$\begin{aligned}
\Phi^{t+1} &= \sum_{i \in C} W_i^{t+1} \overset{(10)}{=} \sum_{i \in C} W_i^t e^{\epsilon \ell_i^t}, \\
&\overset{(29)}{\leq} \sum_{i \in C} W_i^t (1 + \epsilon \ell_i^t + \epsilon^2 \ell_i^t)^2), \qquad e^x \leq 1 + x + x^2, \forall x \in [0,1] \\
&= \Phi^t \sum_{i \in C} p_i^t (1 + \epsilon \ell_i^t + \epsilon^2 (\ell_i^t)^2), \qquad p_i^t = \frac{W_i^t}{\sum_{j \in C} W_j^t} = \frac{W_i^t}{\Phi^t} \\
&= \Phi^t (1 + \epsilon \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle + \epsilon^2 \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle), \qquad (\boldsymbol{\ell}^t)^2 = [(\ell_i^t)^2]_{i \in C} \\
&\leq \Phi^t e^{\epsilon \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle + \epsilon^2 \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle}, \qquad 1 + x \leq e^x.
\end{aligned} \tag{30}$$

So, at round T,

$$\begin{aligned}
e^{\epsilon \sum_{t=1}^{T} \ell_{i^*}^t} = W_{i^*}^T \leq \Phi^T &\leq \Phi^0 e^{\epsilon \sum_{t=1}^{T} \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle + \epsilon^2 \sum_{t=1}^{T} \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle}, \\
\epsilon \sum_{t=1}^{T} \ell_{i^*}^t &\leq \ln|C| + \epsilon \sum_{t=1}^{T} \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle + \epsilon^2 \sum_{t=1}^{T} \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle, \quad \text{take logarithm} \\
\sum_{t=1}^{T} \ell_{i^*}^t - \sum_{t=1}^{T} \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle &\leq \frac{\ln|C|}{\epsilon} + \epsilon \sum_{t=1}^{T} \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle, \qquad \text{divided by } \epsilon \text{ and rearrange.}
\end{aligned} \tag{31}$$

Thus,

$$\begin{aligned}
R_T &= \sum_{t=1}^{T} \ell_{i^*}^t - \mathbb{E}[\sum_{t=1}^{T} \ell_{i^t}^t] = \sum_{t=1}^{T} \ell_{i^*}^t - \sum_{t=1}^{T} \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle \\
&\overset{(31)}{\leq} \frac{\ln|C|}{\epsilon} + \epsilon \sum_{t=1}^{T} \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle \leq \frac{\ln|C|}{\epsilon} + \epsilon \bar{L}^2 T, \quad \boldsymbol{p}^t \text{ is probability, and } \ell_i^t \in [0, \bar{L}].
\end{aligned} \tag{32}$$

The latter inequality yields $R_T \leq 2\bar{L}\sqrt{T \log|C|}$ as $\epsilon = \frac{1}{\bar{L}}\sqrt{\frac{\ln|C|}{T}}$. □

C PROOF OF THEOREM 4.1

We first introduce some auxiliary lemmas to describe the properties of reward vectors. For any set $\tilde{A} \subseteq \mathcal{S} \times C \times [M]$ and given color $\boldsymbol{m} = [m_s]_{s \in \mathcal{S}}$ at round t, let $\overline{F}^t(\tilde{A}, \boldsymbol{m}) = f^t(\texttt{sample}_{\boldsymbol{m}}(\tilde{A}))$. Let $\boldsymbol{m}^t$

be the vector of colors at the beginning of round t. Let also

$$i^t_{s,m} = \begin{cases} \text{the item returned by } \mathcal{E}_{s,m}.\text{arm() the last time it was called (including } t\text{), or} \\ \text{an arbitrary item if the selector has never been called.} \end{cases} \tag{33}$$

Note that, at time t, the selector $\mathcal{E}_{s,m}$.arm() is indeed called for all slots s on a path of a request $r \in \mathcal{R}^t$ when $m = m^t_s$. Let $\tilde{A}^t \subseteq \mathcal{S} \times C \times [M]$ be the triplet set constructed by Alg. 2 at round t, i.e., the set comprising triplets

$$(s, i^t_{s,m}, m) \text{ for all } s \in \mathcal{S} \text{ and } m \in [M].$$

Note that such triplets are updated at all slots in paths of requests in timeslot t; all other triplets remain unaltered. We impose an ordering over all such triplets, defined by an ordering over colors first and slots second (the latter given by Eq. (1)). Under this ordering, similar to $\tilde{A}^-_{s,m}$ defined before Thm. A.1, let $\tilde{A}^{t-}_{s,m}$ equal $\tilde{A}^t$ just "before" $(s, i^t_{s,m}, m)$ is added at round t; this addition is conceptual, presuming these triplets are "added" one-by-one under the aforementioned ordering to construct $\tilde{A}^t$. Under this convention,

$$\tilde{A}^{t-}_{s,m} = \{(s', i^t_{s',m}, m) : s' \in \mathcal{S}, m' < m\} \cup \{(s', i^t_{(s',m)}, m) : s' \prec s\}.$$

Lemma C.1. *At round t, for all storage slot $s \in \bigcup_{(i,p)\in\mathcal{R}^t} \mathcal{S}_p$, the reward vector computed by Eq. (15), i.e., the vector $\boldsymbol{\ell}^r(s, m^t_s) \in \mathbb{R}^{|C|}_+$ with coordinates:*

$$\ell^r_{i'}(s, m^t_s) = \begin{cases} \max_{(v',j')\in\mathcal{S}_{\leq s}+s} w^p_{v'}, & i' = i \\ \max_{(v',j')\in\mathcal{S}_{\leq s}} w^p_{v'}, & o.w \end{cases}$$

where $r = (i, p) \in \mathcal{R}^t$ and $\mathcal{S}_{\leq s} = \{s' \in \mathcal{S}_{i,p} : m_{s'} < m_s \text{ or } m_{s'} = m_s, s' \prec s\}$, satisfies the following property:

$$\sum_{\substack{r=(i,p)\in\mathcal{R}^t: \\ s\in\mathcal{S}_p}} \ell^r_{i'}(s, m^t_s) = \overline{F}^t(\tilde{A}^{t-}_{s,m^t_s} + (s, i', m^t_s), \boldsymbol{m}^t), \text{ for all } i' \in C. \tag{34}$$

Proof. From the definition Eq. (5) of f_r, we have that for $r = (i, p) \in \mathcal{R}^t$:

$$\begin{aligned} f_r(A) &= \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(A \cap \left\{ \bigcup_{k'\in[k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} \neq \emptyset \right) \\ &= \sum_{\substack{k=\min\{k':\ \exists j\in\mathcal{S}_{p_{k'}} \\ \text{s.t. } ((p_{k'},j),i)\in A\}}}^{|p|-1} w_{p_{k+1}p_k} = \max_{v\in p:\exists j \text{ s.t. } ((v,j),i)\in A} w^p_v, \end{aligned} \tag{35}$$

where w^p_v is the cumulative upstream cost defined in Eq. (13). Then,

$$\begin{aligned} \overline{F}^t(\tilde{A}, \boldsymbol{m}) = f^t(\texttt{sample}_{\boldsymbol{m}}(\tilde{A})) &\overset{(22)}{=} \sum_{r\in\mathcal{R}^t} f_r(\{(s, i) \in \mathcal{S} \times C : (s, i, m_s) \in \tilde{A}\}) \\ &\overset{(35)}{=} \sum_{(i,p)\in\mathcal{R}^t} \sum_{\substack{k=\min\{k':\ \exists j\in\mathcal{S}_{p_{k'}} \\ s.t.\ ((p_{k'},j),i,m_s)\in\tilde{A}\}}}^{|p|-1} w_{p_{k+1}p_k} = \sum_{(i,p)\in\mathcal{R}^t} \max_{v\in p:\exists j \text{ s.t. } ((v,j),i,m_{(v,j)})\in\tilde{A}} w^p_v. \end{aligned} \tag{36}$$

Then, for $\tilde{A}' = \tilde{A}^{t-}_{s,m^t_s} + (s, i', m^t_s)$, we have

$$\overline{F}^t(\tilde{A}^{t-}_{s,m^t_s} + (s, i', m^t_s), \boldsymbol{m}^t) \overset{(36)}{=} \sum_{\substack{(i,p)\in\mathcal{R}^t:\\ s\in\mathcal{S}_p}} \max_{((v',j'),i,m_{s'})\in\tilde{A}'} w^p_{v'}$$
$$= \begin{cases} \sum_{\substack{(i,p)\in\mathcal{R}^t:\\ s\in\mathcal{S}_p}} \max_{(v',j')\in\mathcal{S}_{\leq s}+s} w^p_{v'}, & i' = i \\ \sum_{\substack{(i,p)\in\mathcal{R}^t:\\ s\in\mathcal{S}_p}} \max_{(v',j')\in\mathcal{S}_{\leq s}} w^p_{v'}, & o.w \end{cases} = \sum_{\substack{r=(i,p)\in\mathcal{R}^t:\\ s\in\mathcal{S}_p}} \ell^r_{i'}(s, m^t_s). \tag{37}$$

The second equality holds because of the definition of accumulate weights by Eq. (13) and the fact that $v = \arg\min_v k_p(v) = \arg\max_v w^p_v$. The Third equality holds by the definitions of $\tilde{A}'$ and $\mathcal{S}_{\leq s}$. □

If the requests in $\mathcal{R}^t$ do not cross, it behaves same as the one request scenario. If requests do cross each other, the reward vector calculated by storage slot (v, j) is the summation of separate reward vectors deriving from each request $r^t \in \mathcal{R}^t$. Actually, it is equivalent to calculating the reward vector and calling operation feedback($\boldsymbol{\ell}^t(s, m_s)$) separately when each request arrives. We prove the above statement by the following lemma:

Lemma C.2. *Calling feedback() with reward vector $\sum_{i=1}^k \boldsymbol{\ell}_i$ is equivalent to a sequence of k feedback calls, with reward vectors $\boldsymbol{\ell}_i$.*

Proof. If we feedback a reward: $\sum_i \boldsymbol{\ell}_i$, the weight vector in it is: $\forall i \in C$, $W^{t+1}_i = W^t_i e^{\epsilon \sum_i \ell_i}$. If we feedback rewards $\boldsymbol{\ell}_i$ for all i separately, in the end, the weight vector in it is: $\forall i \in C$, $W^{t+1}_i = W^t_i \prod_i e^{\epsilon \ell_i}$. These two feedback scenario lead to same state in hedge selector. □

Lemma C.3. *At round t, given selected color $\boldsymbol{m}$, for $s \notin \bigcup_{(i,p)\in\mathcal{R}^t} \mathcal{S}_p$ or $m \neq m_s$, all $i', j' \in C$, $\overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}) = \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, j', m), \boldsymbol{m})$.*

Proof. When $s \notin \bigcup_{(i,p)\in\mathcal{R}^t} \mathcal{S}_p$ or $m \neq m_s$, according to Eq. (37), for all $i' \in C$,

$$\overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}) = \sum_{(i,v)\in\mathcal{R}^t} \max_{((v',j'),i,m_{s'})\in\tilde{A}'} w^p_{v'} = \sum_{(i,v)\in\mathcal{R}^t} \max_{(v',j')\in\mathcal{S}_{\leq s}} w^p_{v'}. \tag{38}$$

□

Finally, we can prove Theorem 4.1.

Proof. For all $s \in \mathcal{S}, m \in [M]$, we denote by $\mathcal{T}_{s,m}$ be the set of rounds where hedge selector $\mathcal{E}_{s,m}$ receives reward vector in our algorithm in T rounds, i.e., $\mathcal{T}_{s,m} = \{t \in [T] : v \in p^t, m = m^t_s\}$. For any $s \in \mathcal{S}, m \in [M]$, and $i' \in C$, since hedge selectors are no-regret algorithms, let $R^T_{s,m}$ be the regret of $\mathcal{E}_{s,m}$ during rounds $\mathcal{T}_{s,m}$. We denote $l^t_i(s, m)$ the i-th coordinate of total reward vector for hedge selector $\mathcal{E}_{s,m}$ at round t, i.e., $l^t_i(s, m) = \sum_{\substack{r=(i,p)\in\mathcal{R}^t:\\ s\in\mathcal{S}_p}} \ell^r_i(s, m)$. According to the definition of regret in Eq. (27), we have

$$R^T_{s,m} = \sum_{t\in\mathcal{T}_{s,m}} \ell^t_{i^*}(s, m) - \sum_{t\in\mathcal{T}_{s,m}} \ell^t_{i^t_{s,m}}(s, m) \geq \sum_{t\in\mathcal{T}_{s,m}} \ell^t_{i'}(s, m) - \sum_{t\in\mathcal{T}_{s,m}} \ell^t_{i^t_{s,m}}(s, m), \tag{39}$$

where i^* is the best selection in hindsight, i.e., $i^* = \arg\max_{i\in C} \sum_{t\in\mathcal{T}_{s,m}} \ell^t_i$. Then, by Lemma C.1,

$$\sum_{t\in\mathcal{T}_{s,m}} \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i^t_{s,m}, m), \boldsymbol{m}) \geq \left(\sum_{t\in\mathcal{T}_{s,m}} \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}) \right) - R^T_{s,m}. \tag{40}$$

For $t \in [T] \setminus \mathcal{T}_{s,m}$, all $s \in \mathcal{S}, m \in [M], i' \in C$, by Lemma C.3,

$$\overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i^t_{s,m}, m), \boldsymbol{m}) = \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}). \tag{41}$$

Thus, for all $s \in \mathcal{S}, m \in [M], i' \in C$,

$$\sum_{t=1}^{T} \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i^t_{s,m}, m), \boldsymbol{m}) \geq \left(\sum_{t=1}^{T} \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}) \right) - R^T_{s,m}. \tag{42}$$

Taking the expectation of both sides over m, and over $i^t_{s,m}$ and choosing i to maximize the right hand side, we get

$$\sum_{t=1}^{T} F^t(\tilde{A}^{t-}_{s,m} + (s, i^t_{s,m}, m)\}) \geq \max_{i \in C} \left(\sum_{t=1}^{T} F^t(\tilde{A}^{t-}_{s,m} + (s, i', m)) \right) - \epsilon_{s,m}, \tag{43}$$

where we define $F^t(\tilde{A}) = \mathbb{E}_{\boldsymbol{m}}[\mathbb{E}_{i^t_{s,m}}[f^t(\texttt{sample}_{\boldsymbol{m}}(\tilde{A}))]]$ and $\epsilon_{s,m} = \mathbb{E}[R^T_{s,m}]$.

We now define some additional notation. For any set $\vec{A}$ of vector in $\mathcal{S} \times C^T$, define

$$f(\vec{A}) = \sum_{t=1}^{T} f^t(\{(s, i^t) : (s, \vec{i}) \in \vec{A}\}), \tag{44}$$

where $\vec{i} = [i^t]_{t=1}^T$. Next, for any set $\vec{\tilde{A}} \subseteq \mathcal{S} \times C^T \times [M]$, and given $\boldsymbol{m} = [m_s]_{s \in \mathcal{S}}$, define $\texttt{sample}_{\boldsymbol{m}}(\vec{\tilde{A}}) = \{(s, \vec{i}) \in \{s\} \times C^T : (s, \vec{i}, m_s) \in \vec{\tilde{A}}\}$. Define $F(\vec{\tilde{A}}) = \mathbb{E}_{\boldsymbol{m}}[\mathbb{E}_{\vec{i}_{s,m}}[f(\texttt{sample}_{\boldsymbol{m}}(\vec{\tilde{A}}))]]$, where $\vec{i}_{s,m} = [i^t_{s,m}]_{t=1}^T \in C^T$. By linearity of expectation,

$$F(\vec{\tilde{A}}) = \sum_{t=1}^{T} F^t(\{(s, i^t, m) : (s, \vec{i}, m) \in \vec{\tilde{A}}\}). \tag{45}$$

Analogously to $\tilde{A}^{t-}_{s,m}$, define $\vec{\tilde{A}}^{-}_{s,m} = \{(s', \vec{i}_{s',m'}, m') : s' \in \mathcal{S}, m' < m\} \cup \{(s', \vec{i}_{s',m}, m) : s' \prec s\}$. Thus, for any $(s, \vec{i}, m) \in \mathcal{S} \times C^T \times [M]$, we have $F(\vec{\tilde{A}}^{-}_{s,m} + (s, \vec{i}, m)) = \sum_{t=1}^{T} F^t(\tilde{A}^{t-}_{s,m} + (s, i_t, m))$. Combining this with (43) , we get:

$$F(\vec{\tilde{A}}^{-}_{s,m} + (s, \vec{i}_{s,m}, m)) \geq \max_{i' \in C} \left(F(\vec{\tilde{A}}^{-}_{s,m} + (s, [i']^T, m)) \right) - \epsilon_{s,m}. \tag{46}$$

Having proved (46), we can now use Thm. A.1 to complete the proof. Define a new partition matroid over ground set $\{\mathcal{S} \times C^T\}$ with feasible solution $\vec{\mathcal{D}} := \{\vec{A} \subset \mathcal{S} \times C^T : |\vec{A} \cap (\{s\} \times C^T)| = 1, \forall s \in \mathcal{S}\}$. Let $\vec{\tilde{A}} = \{(s, \vec{i}_{s,m}, m) : s \in \mathcal{S}, m \in [M]\}$. It is easy to verify that F^t is monotone submodular, and F is also monotone submodular by linearity. Thus, by Thm. A.1,

$$F(\vec{\tilde{A}}) \geq \beta(M, |\mathcal{S}|) \cdot \max_{\vec{A} \in \vec{\mathcal{D}}} \{f(\vec{A})\} - \sum_{s \in \mathcal{S}} \sum_{m=1}^{M} \epsilon_{s,m}. \tag{47}$$

By definition, $F(\vec{\tilde{A}}) = \mathbb{E}[\sum_{t=1}^{T} f^t(A^t)]$, and $\max_{\vec{A} \in \vec{\mathcal{D}}} \{f(\vec{A})\} \geq \max_{A \in \mathcal{D}} \{\sum_{t=1}^{T} f^t(A)\}$, we get

$$\mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t)\right] \geq \beta(|\mathcal{S}|, M) \cdot \max_{A \in \mathcal{D}} \left\{ \sum_{t=1}^{T} f^t(A) \right\} - \sum_{s \in \mathcal{S}} \sum_{m=1}^{M} \epsilon_{s,m}. \tag{48}$$

According to Lemma B.1, $\epsilon_{s,m} = \mathbb{E}[R^T_{s,m}] \leq 2\bar{R}\bar{L}\sqrt{|\mathcal{T}_{s,m}| \log |C|} \leq 2\bar{R}\bar{L}\sqrt{T \log |C|}$, then

$$\mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t)\right] \geq \beta(|\mathcal{S}|, M) \cdot \max_{A \in \mathcal{D}} \left\{ \sum_{t=1}^{T} f^t(A) \right\} - 2\bar{R}\bar{L}|\mathcal{S}|M\sqrt{T \log |C|}. \tag{49}$$

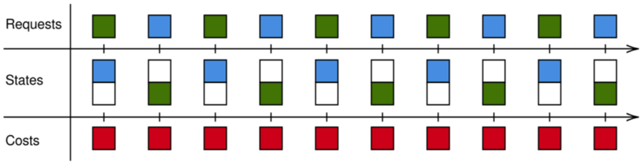


Fig. 10. A simplified instance of GRD with linear regret. The cache can only store a single file, and at any given moment the adversary requests the file that is not stored in the cache. GRD updates its state greedily to store the requested file and oscillates between the two possible states. The optimal static strategy stores one of the files permanently incurring a total cost of $T/2$, while GRD incurs a total cost T.

□

D PROOF OF LEMMA 5.1

Proof. Consider a cache network of two nodes u and v and a catalog of two files represented by the set $\{1, 2\}$. The cache node u has capacity 1 and the node v is a repository node containing the files 1 and 2. The hedge selector is initialized with a distribution of the possible states $\boldsymbol{p}^1 = (1/2, 1/2)$. According to Alg. 1, the hedge selector adds $\epsilon = \Theta\left(\frac{1}{\sqrt{T}}\right)$ fraction to the component corresponding to the requested file, and reduces the same quantity from the other component. When the requested files sequence is $\{1, 2, 1, 2, \dots\}$, this gives the following distributions:

$$\{\boldsymbol{p}^1, \boldsymbol{p}^2, \boldsymbol{p}^3, \boldsymbol{p}^4, \dots\} = \{(1/2, 1/2), (1/2 + \epsilon, 1/2 - \epsilon), (1/2, 1/2), (1/2 + \epsilon, 1/2 - \epsilon), \dots\}.$$

The distribution $\boldsymbol{p}^1$, gives two integral states $(1, 0)$ w.p. $1/2$ and $(0, 1)$ w.p. $1/2$, and $\boldsymbol{p}^2$ can give two integral states $(1, 0)$ w.p $1/2 + \epsilon$ and $(0, 1)$ w.p. $1/2 - \epsilon$. The expected update cost experienced in expectation from $t = 1$ to $t = 2$ is:

$$\mathbb{E}\left[\mathrm{UC}(A^1, A^2)\right] = 1/2\,(1/2 - \epsilon) + 1/2\,(1/2 + \epsilon) = 1/2.$$

The decomposition of $\boldsymbol{p}^3$ is the same as $\boldsymbol{p}^1$. The update cost experienced in expectation from $t = 2$ to $t = 3$ is:

$$\mathbb{E}\left[\mathrm{UC}(A^2, A^3)\right] = (1/2 - \epsilon)\,1/2 + (1/2 + \epsilon)\,1/2 = 1/2.$$

The sequence repeats and the same costs are obtained. The total update cost is:

$$\mathbb{E}\left[\sum_{t=1}^{T} \mathrm{UC}(A^t, A^{t+1})\right] = \sum_{t=1}^{T} \mathbb{E}\left[\mathrm{UC}(A^t, A^{t+1})\right] = \frac{T}{2} \tag{50}$$

This is an update cost of $\Omega(T)$ paid in expectation.

□

E PROOF OF LEMMA 4.3

Proof. Assume a cache network formed of a designated server v and a cache with storage capacity $c_u \in \mathbb{N}$. The catalog C contains $2c_u$ items and, without lack of generality we assume that cache u initially contains the set of items $\{c_u + 1, \dots, 2c_u\}$. Requests arrive only at node u, and we can identify a request with the requested item because there is only one possible path $(\{u, v\})$. The forwarding cost between u and v is $w \in \mathbb{R}_+$.

We consider request sequences with one request every Δ time units and one request per round $(\bar{R} = 1)$ and we denote by i_t he item requested at time t. Moreover, for simplicity, we assume the time horizon T to be proportional to $2c_u$ $(T = m \times 2c_u)$. The service cost without caching is wT.

The policy GRD maintains a vector $\boldsymbol{z}_t$ and updates it after every request as follows [41, Eq. (22)]:

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t e^{-\Delta\beta} + w\beta\boldsymbol{e}_{i_t}, \tag{51}$$

where $\boldsymbol{e}_{i_t} = \left[\mathbb{1}_{\{i=i_t\}}\right]_{i\in C}$ and $\boldsymbol{z}_0 = \boldsymbol{0}$. After the request time t GRD stores in the cache the c_u items in C that correspond to the largest c_u components of $\boldsymbol{z}_{t+1}$ at time t. Consider the request sequence

$$\{1, 2, \ldots, 2c_u, 1, 2, \ldots, 2c_u, \ldots, 1, 2, \ldots, 2c_u\}. \tag{52}$$

Under this request sequence, GRD behaves as LRU and simply stores at any time the c_u most recently requested items. In fact, item j is requested at time instants $hT + j$ for $h \in \{0, 1, \ldots, m-1\}$. At time $t = kT + i$, item j has been requested for the last time $(i-j) \mod 2c_u$ time instants earlier. The corresponding component of the vector $\boldsymbol{z}_{t+1}$ has the value

$$(\boldsymbol{z}_{t+1})_j = e^{-\beta\Delta((i-j) \mod 2c_u)} \times \sum_{\substack{h\in\{0,1,\ldots,m-1\},\\ hT+j\leq kT+i}} e^{-\beta\Delta h}.$$

The maximum value is achieved for $j = i$. The component becomes progressively smaller as j decreases from i to 1, because the first term in the product becomes smaller while the second terms does not change. It keeps decreasing as j decreases from $2c_u$ to $i+1$, not only because the first term decreases, but also because the second term decreases as less addends are considered in the sum.

As GRD behaves as LRU, when a new request i_t arrives at node u, it is never found in the cache. GRD incurs then a total service cost wT and null caching gain. At the same time, the caching gain of any cache allocation A (with c_u different items stored at u) is $\frac{w}{2}T$, because it is able to serve half of the requests. Then, the α-regret of GRD is at least equal to $\alpha\frac{wT}{2}$. A simplified instance of GRD is shown in Fig. 10 to provide some intuition of our proof.

□

F PROOF OF THEOREM 5.2

We begin by giving some intuition behind our approach. The hedge selector in Alg. 1 effectively maintains a distribution $\boldsymbol{p}^t = \left[\frac{W_i^t}{\sum_{j\in C} W_j^t}\right]_{i\in C}$ for every round t. The randomized action i_t taken at time t by the selector always satisfies $\mathbb{E}[\boldsymbol{e}_{i^t}] = \boldsymbol{p}^t$ by definition, where $\boldsymbol{e}_i$ is the i-th basis vector. Consider for instance, that the rewards given to the hedge selector are always uniform. Since each action is equally important, then the distribution $\boldsymbol{p}^t$ is fixed and it is the uniform distribution. Thus, the hedge selector controlling item placements, will evict and fetch a new content with probability $1 - \frac{1}{|C|^2}$ at each time step. Clearly, since the distribution is fixed for every time step, then these movements are unnecessary. An optimal strategy in this scenario is to pick an action u.a.r at the start and stick with it.

We generalize this concept by taking minimal probabilistic jumps to a new state, only when it is necessary to maintain a change of distribution from $\boldsymbol{p}^t$ to $\boldsymbol{p}^{t+1}$. This concept is known in the literature as optimal transport or the earth mover distance [67]. The objective is to transport probability mass from a distribution to another, while minimizing the associated metric. In this scenario, it corresponds to a minimum-cost flow problem. We propose an iterative algorithm that builds the optimal flow (joint distribution). By building a feasible flow at time t from $\boldsymbol{p}^t$ to $\boldsymbol{p}^{t+1}$. Then, the algorithm takes elementary steps that generates a sequence of random variables whose marginal distribution is progressively closer to $\boldsymbol{p}^{t+1}$

The proof is split in multiple parts. We first introduce Lemma F.1 that links the hedge selector update rule to online mirror descent [13]. This allows us to use convex optimization techniques to provide the proof of Lemma F.2, that gives a family of coupling schemes with sublinear update cost.

In section F.3, we dissect the hedge selector and bound the update cost of its two components, the *coupled_movement* and *elementary_δmovement* subroutines.

In some parts of the proof we switch to vector notation rather than the set notation for allocations. For any allocation $A \in \mathcal{S} \times C$ the corresponding allocation vector is denoted by $\boldsymbol{x} = [\mathbb{1}_{\{(s,i)\in A\}}]_{(s,i)\in\mathcal{S}\times C}$, and $\boldsymbol{x}_s = [\mathbb{1}_{\{(s,i)\in A\}}]_{(s,i):i\in C}$. The weighted l_1 norm is defined as:

$$||\boldsymbol{x}_s||_{1,\boldsymbol{w}'} := \sum_{i\in C} w'_{s,i}|x_{s,i}|. \tag{53}$$

With slight abuse of notation, we consider that for any $s \in \mathcal{S}$:

$$\begin{aligned}\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1}) &:= \sum_{i\in C} w'_i \max(0, x_i^{t+1} - x_i^t) & (54)\\ &= \mathrm{UC}(A^t \cap \{(s,i) : i \in C\}, A^{t+1} \cap \{(s,i) : i \in C\}). & (55)\end{aligned}$$

Also, note that $\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1}) \leq ||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}$.

F.1 Auxiliary Lemma

We start by introducing an auxiliary lemma.

Lemma F.1. *$\mathcal{E}$.feedback($\boldsymbol{\ell}$) for the hedge selector in Alg. 1 updates its internal weights W_t to W_{t+1} equivalently as $\nabla\Phi(\boldsymbol{W}^{t+1}) = \nabla\Phi(\boldsymbol{W}^t) + \epsilon\boldsymbol{\ell}^t$ where $\Phi(\boldsymbol{W}) = \sum_{i\in C} W_i \log(W_i)$ and $\epsilon \in \mathbb{R}_+$ is the step size.*

Proof. We know that

$$\frac{\partial\Phi(\boldsymbol{W})}{\partial W_i} = 1 + \log(W_i) \tag{56}$$

From $\nabla\Phi(\boldsymbol{W}^{t+1}) = \nabla\Phi(\boldsymbol{W}^t) + \epsilon\boldsymbol{\ell}^t$ we have:

$$1 + \log(W_i^{t+1}) = 1 + \log(W_i^t) + \epsilon\ell_i^t, \tag{57}$$

then,

$$W_i^{t+1} = W_i^t e^{\epsilon\ell_i^t}, \tag{58}$$

which is exactly $\mathcal{E}$.feedback($\boldsymbol{\ell}$) □

F.2 Family of Coupling Schemes with Sublinear Update Cost

The following Lemma provides a sufficient condition on the joint distribution of $(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})$ (the family of coupling schemes), that leads to sublinear update cost for DistributedTGOnline.

Lemma F.2. *Consider a hedge selector, shown in Alg. 1, and a joint distribution of $(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})$ that satisfies for all $t \in [T-1]$:*

(1) $\mathbb{E}[\boldsymbol{x}_s^t] = \boldsymbol{p}^t$ and $\mathbb{E}[\boldsymbol{x}_s^{t+1}] = \boldsymbol{p}^{t+1}$.
(2) $\mathbb{E}[||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}] = O(||\boldsymbol{p}^{t+1} - \boldsymbol{p}^t||_{1,\boldsymbol{w}'})$.

This algorithm incurs an expected update cost of the same order of the update cost of probabilities. Selecting $\epsilon = \Theta(\frac{1}{\sqrt{T}})$, $K = \Omega(T)$, gives a $O(\bar{R}\bar{L}\sqrt{T})$ expected update cost.

Proof. We know $\Phi(\boldsymbol{W}) = \sum_{i\in C} W_i \log(W_i)$ is 1-strong convex w.r.t. $||\cdot||_1$ on the simplex $\triangle_{|C|} = \{\boldsymbol{p} \in \mathbb{R}_+^{|C|} : \sum_{i\in C} p_i = 1\}$ [13]. Thus, $\Phi(x) - \Phi(y) \leq \nabla\Phi(x)^\top(x-y) - \frac{1}{2}||x-y||_1^2$.

Then, we will prove the Lemma F.2. Recall that, as shown in Alg. 2, at every K times, each storage slot will choose a color m_s^t uniformly at random from $[M]$. At every rounds, the corresponding hedge selector $\mathcal{E}_{s,m_s^t}$ will be fed a reward vector $\boldsymbol{\ell}^t(s, m_s^t)$, and call $\mathcal{E}$.feedback($\boldsymbol{\ell}$) to update its weight vector. In the following proof, we assume that at round t and $t+1$, color m_s^t and m_s^{t+1} are chosen.

The weight vector $\boldsymbol{W}^t$ is the weight vector maintained by hedge selector $\mathcal{E}_{s,m_s^t}$. The probability $\boldsymbol{p}^t$ is the normalized $\boldsymbol{W}^t$. With the assumptions in Lemma. F.2, there exist constants $\alpha, \beta, \gamma > 0$, such that:

$$\mathbb{E}[\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})] \leq \mathbb{E}[||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}] \leq \alpha||\boldsymbol{p}^{t+1} - \boldsymbol{p}^t||_{1,\boldsymbol{w}'} \leq \beta||\boldsymbol{p}^{t+1} - \boldsymbol{p}^t|| \leq \gamma||\boldsymbol{W}^{t+1} - \boldsymbol{W}^t||, \quad (59)$$

where $||\cdot||$ here is $l1$ norm, the second to last inequality holds because norms can bound each other, and the last inequality holds because the inexpensive property of projection. For round t without color update:

$$\begin{aligned}
&\mathbb{E}[||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}] \leq \gamma||\boldsymbol{W}^{t+1} - \boldsymbol{W}^t|| \\
\leq&\gamma\sqrt{2(\Phi(\boldsymbol{W}^t) - \Phi(\boldsymbol{W}^{t+1}) + \nabla\Phi(\boldsymbol{W}^{t+1})^\top(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t))}, \qquad \text{1-strong convexity} \\
\leq&\gamma\sqrt{2}\sqrt{\Phi(\boldsymbol{W}^t) - \Phi(\boldsymbol{W}^{t+1}) - \nabla\Phi(\boldsymbol{W}^t)^\top(\boldsymbol{W}^t - \boldsymbol{W}^{t+1}) + (\nabla\Phi(\boldsymbol{W}^{t+1}) - \nabla\Phi(\boldsymbol{W}^t))^\top(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t)}, \\
\leq&\gamma\sqrt{2}\sqrt{-\frac{1}{2}||\boldsymbol{W}^t - \boldsymbol{W}^{t+1}||^2 + (\nabla\Phi(\boldsymbol{W}^{t+1}) - \nabla\Phi(\boldsymbol{W}^t))^\top(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t)}, \qquad \text{1-strong convexity} \\
\leq&\gamma\sqrt{2}\sqrt{-\frac{1}{2}||\boldsymbol{W}^t - \boldsymbol{W}^{t+1}||^2 + \epsilon\bar{R}(\boldsymbol{\ell}^t)^\top(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t)}, \qquad \text{Lemma F.1} \\
\leq&\gamma\sqrt{2}\sqrt{-\frac{1}{2}||\boldsymbol{W}^t - \boldsymbol{W}^{t+1}||^2 + \epsilon\bar{R}||\boldsymbol{\ell}^t||_\infty \cdot ||\boldsymbol{W}^{t+1} - \boldsymbol{W}^t||}, \qquad \text{Cauchy–Schwarz inequality} \\
\leq&\gamma\sqrt{2}\sqrt{\frac{(\epsilon\bar{R}||\boldsymbol{\ell}^t||_\infty)^2}{2}}, \qquad az - bz^2 \leq \frac{a^2}{4b}, a, b > 0 \\
=&\gamma\epsilon\bar{R}||\boldsymbol{\ell}^t||_\infty.
\end{aligned}$$

For round t with color update, the cache update cost is bounded by the most expensive cache update, i.e.,:

$$\mathbb{E}[||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}] \leq ||\boldsymbol{w}'||_\infty. \quad (60)$$

The total update cost experienced for the hedge selector associated with slot s:

$$\sum_{t=1}^{T-1}\mathbb{E}[\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})] \leq \sum_{t=1}^{T-1}\gamma\frac{\epsilon\bar{R}||\boldsymbol{\ell}^t||_\infty}{\rho} + \sum_{t=K,2K,\ldots}||\boldsymbol{w}'||_\infty \leq \gamma\epsilon\bar{R}\bar{L}T + \frac{T}{K}||w'||_\infty, \quad (61)$$

where $\bar{L} = \max_{(i,p)\in\mathcal{R}}\{\sum_{k=1}^{|p|-1} w_{p_{k+1}p_k}\} \geq \max_{t\leq T}\{||\boldsymbol{\ell}^t||_\infty\}$. Assume that $K = \Omega(\sqrt{T})$, then $\exists c' > 0$,

$$\sum_{t=1}^{T-1}\mathbb{E}[\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})] \leq \gamma\epsilon R\bar{L}T + \frac{||\boldsymbol{w}||_\infty'}{c'}\sqrt{T}. \quad (62)$$

In order to keep the hedge selectors regret sublinear $O(\sqrt{T})$ with $\epsilon = \Theta(\frac{1}{\sqrt{T}})$. The update cost for all the slots:

$$\sum_{t=1}^{T-1}\sum_{s\in\mathcal{S}}\mathbb{E}[\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})] \leq |\mathcal{S}|\gamma R\bar{L}\sqrt{T} + |\mathcal{S}|\frac{||\boldsymbol{w}||_\infty'}{c'}\sqrt{T}. \quad (63)$$

□

F.3 Dissection of the Coupled Hedge Selector

Assume at round t, storage slot s, item allocation $\boldsymbol{x}_s^t$ with $\mathbb{E}[\boldsymbol{x}_s^t] = \boldsymbol{p}^t$ has a particular allocation $\boldsymbol{x}_s^t$. We introduce the coupling scheme modification to the hedge selector in Alg.1 given in Alg. 3, which could produce $\boldsymbol{x}_s^{t+1}$ satisfying conditions in Lemma. F.2. We first provide expected update cost of an elementary_δmovement subroutine call.

Lemma F.3. *The elementary_δmovement subroutine outputs a random integral cache configuration X' with $\mathbb{E}[\boldsymbol{e}_{X'}] = \boldsymbol{p} - \delta\boldsymbol{e}_i + \delta\boldsymbol{e}_j$. If its input is sampled from a random variable X with $\mathbb{E}[\boldsymbol{e}_X] = \boldsymbol{p}$, then:*

$$\mathbb{E}\left[||\boldsymbol{e}_X - \boldsymbol{e}_{X'}||_{1,\boldsymbol{w}'}\right] = \delta(w'_{s,j} + w'_{s,i}). \tag{64}$$

Proof.

First part. Showing that $\mathbb{E}\left[\boldsymbol{e}_{X'}\right] = \boldsymbol{p} - \delta\boldsymbol{e}_i + \delta\boldsymbol{e}_j$.

$$\begin{aligned}
\mathbb{E}\left[\boldsymbol{e}_{X'}\right] &= \sum_{l\in C\setminus\{i\}} p_l \mathbb{E}\left[\boldsymbol{e}_{X'}|X = l\right] + p_i \mathbb{E}\left[\boldsymbol{e}_{X'}|X = i\right] && (65)\\
&= \sum_{l\in C\setminus\{i\}} p_l \boldsymbol{e}_l + p_i \mathbb{E}\left[\boldsymbol{e}_{X'}|X = i\right], && \text{Line 26, Algorithm 3} \quad (66)\\
&= \sum_{l\in C\setminus\{i\}} p_l \boldsymbol{e}_l + p_i \left(\frac{p_i - \delta}{p_i}\boldsymbol{e}_i + \frac{\delta}{p_i}\boldsymbol{e}_j\right), && \text{Line 23, Algorithm 3} \quad (67)\\
&= \sum_{l\in C} p_l \boldsymbol{e}_l - \delta\boldsymbol{e}_i + \delta\boldsymbol{e}_j = \boldsymbol{p} - \delta\boldsymbol{e}_i + \delta\boldsymbol{e}_j. && (68)
\end{aligned}$$

Second part. The only movement that can be caused by running the subroutine is at line 23, given that $X = i$ with probability p_i, we replace this value by j with probability $\frac{\delta}{p_i}$. Hence, the expected update cost is given by:

$$\mathbb{E}[||\boldsymbol{e}_X - \boldsymbol{e}_{X'}||_{1,\boldsymbol{w}'}] = \frac{p_i\delta}{p_i}(w'_{s,j} + w'_{s,i}) = \delta(w'_{s,j} + w'_{s,i}). \tag{69}$$

□

We now introduce lemma F.4 providing the expected update cost of a coupled_movement subroutine call.

Lemma F.4. *If the input to coupled_movement subroutine in Alg.3 is i_s^t with $\mathbb{E}[\boldsymbol{e}_{i_s^t}] = \boldsymbol{p}^t$, then it outputs an item i_s^{t+1}, where $\mathbb{E}[\boldsymbol{e}_{i_s^{t+1}}] = \boldsymbol{p}^{t+1}$, and $\mathbb{E}_{i_s^t}[\mathbb{E}_{i_s^{t+1}|i_s^t}[||\boldsymbol{e}_{i_s^t} - \boldsymbol{e}_{i_s^{t+1}}||_{1,\boldsymbol{w}'}]] = ||\boldsymbol{p}^t - \boldsymbol{p}^{t+1}||_{1,\boldsymbol{w}'}$.*

Proof. The distribution over the catalog changes from a fractional state $\boldsymbol{p}_t \in \Delta_{|C|}$ to $\boldsymbol{p}^{t+1} \in \Delta_{|C|}$. The set $I = \left\{i \in C : x_{t+1,i} - x_{t,i} > 0\right\}$ in line 1 of Algorithm 3 is the set of components that have a fractional increase, then we get:

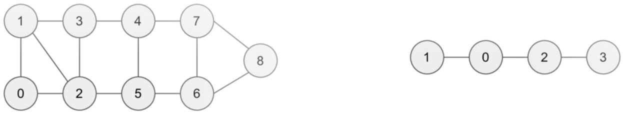
$$\boldsymbol{p}^{t+1} = \boldsymbol{p}^t + \sum_{j\in I} m_j e_j - \sum_{i\in C\setminus I} m_i e_i, \tag{70}$$

where $m_i, i \in C$ is the absolute fractional change in component i of the cache. The fractional update cost is the following:

$$||\boldsymbol{p}^t - \boldsymbol{p}^{t+1}||_{1,\boldsymbol{w}'} = \sum_{j\in I} m_j w'_{s,j} + \sum_{j\in C\setminus I} m_i w'_{s,i}. \tag{71}$$

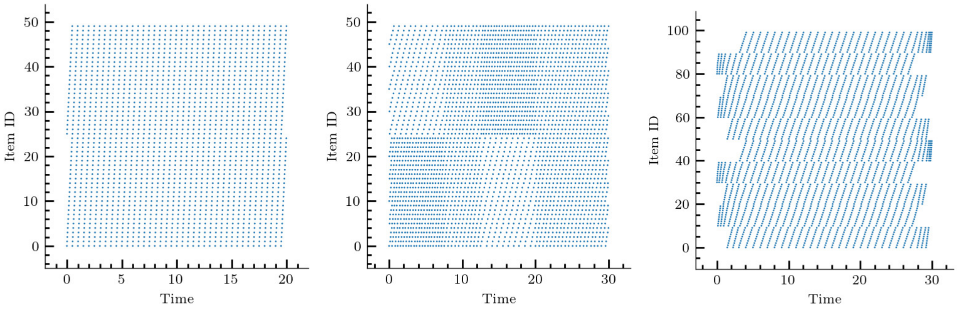
A flow $[\delta_{i,j}]_{(i,j)\in C^2}$ is constructed to transport $\sum_{i\in C\setminus I} m_i$ mass from the components in I to the components in $C \setminus I$ in line 15. The expected value of the allocation generated by output variable i_s^{t+1} is given by:

$$\begin{aligned}
\mathbb{E}[\boldsymbol{e}_{i_s^{t+1}}] &\overset{(68)}{=} \mathbb{E}[\boldsymbol{e}_{i_s^t}] + \sum_{i\in C\setminus I}\sum_{j\in I} \delta_{i,j}(\boldsymbol{e}_j - \boldsymbol{e}_i) && (72)\\
&= \boldsymbol{p}^t - \sum_{i\in C\setminus I} m_i \boldsymbol{e}_i + \sum_{j\in I} m_j \boldsymbol{e}_j = \boldsymbol{p}^{t+1}. && (73)
\end{aligned}$$



(a) `Abilene` topology (b) `Path` topology

Fig. 11. Topologies used for adversarial requests.



(a) *Stationary Adversarial* trace (b) *Sliding Popularity Adversarial* trace (c) *SN Adversarial* trace

Fig. 12. Adversarial Request models for `abilene` and `path`. Each dot indicates an access to an item/ a request.

The expected movements incurred when Algorithm 3 is executed is the following:

$$\mathbb{E}[||\boldsymbol{e}_{i_s^t} - \boldsymbol{e}_{i_s^{t+1}}||_{1,\boldsymbol{w}'}] \overset{(69)}{=} \sum_{i \in C \setminus I} \sum_{j \in I} \delta_{i,j}(w'_{s,j} + w'_{s,i}) = \sum_{j \in I} m_j w'_{s,j} + \sum_{j \in C \setminus I} m_i w'_{s,i}$$

$$\overset{(71)}{=} ||\boldsymbol{p}^t - \boldsymbol{p}^{t+1}||_{1,\boldsymbol{w}'}.$$

□

G ADVERSARIAL INSTANCES

In this section, we provide additional details about the topologies `path` and `abilene`. These are motivated by the proof of Lemma 4.3, using round-robin schemes for which greedy/myopic online algorithms would perform poorly.

G.1 Topology Configuration

The `abilene` and `path` topologies are shown in Fig. 11 (a) and (b) respectively.

Adversarial setup of `abilene`. We set the weight of each edge to be $w = 100$. The query nodes are $\{0, 5\}$. We put a cache at each node with a capacity selected u.a.r from the set $\{0, 1\}$, except for nodes $\{0, 5\}$ that have capacity 5. For every item in the catalog we select its source node u.a.r to be 7 or 8.

Adversarial setup of `path`. We set the weight of each edge to be $w = 100$. The query node is 0. We put two caches at nodes 0 and 1 with capacity 5. The whole catalog is stored at node 1.

G.2 Adversarial Traces

The adversarial traces patterns are shown in Fig. 12. The *Stationary Adversarial* trace is generated using the sequence

$$\{0, 25, 1, 26, \dots, 24, 49, 0, 25, \dots\}. \tag{74}$$

The *Sliding Popularity Adversarial* trace is generated by mixing the two sequences

$$s_1 = \{i \mod 5 + 5k : i \in [100], k \in [5]\}, \tag{75}$$

$$s_2 = \{i \mod 5 + 5k + 25 : i \in [100], k \in [5]\}. \tag{76}$$

We generate requests from s_1 and s_2 starting at the same time, except that we generate requests from s_1 twice as fast as s_2. When we finish generating requests from a sequence we alternate the speed. This is done once, then at the final stage we generate requests with at the same speed. The *SN Adversarial* trace is generated using the same cyclic pattern as in the *Sliding Popularity Adversarial* trace, with the difference that a group of 5 items arrive according to according to a homogeneous Poisson process of rate $\gamma = 1$.

H JOINTLY OPTIMIZING CACHING AND ROUTING

In the extended model by Ioannidis and Yeh [43], a request $r = (i, b) \in \mathcal{R}$ is determined by (a) the item $i \in C$ requested, (b) the source node $b \in V$ of the request. For each request $r = (i, b)$, there exists a set of paths $\mathcal{P}_{(i,b)}$, which the request can follow towards a designated server in $\mathcal{D}_i$. The goal is to jointly determine the content allocation *as well as* the paths that requests follow.

In particular, the *path assignment* is represented by $P = \{p_r\}_{r\in\mathcal{R}} \in \prod_{r\in\mathcal{R}} \mathcal{P}_r$, where $p_r \in \mathcal{P}_r$ indicates that request $r = (i, b) \in \mathcal{R}$ follows path p_r to fetch item i. It is easy, and natural, to write the cost objective in terms of the content allocation A and the routing assignment P. However, to show that it is a submodular assignment problem, with constrains similar to the ones we encounter in caching, we deviate from [43] and express the objective in terms of the *complementary path assignment* $\overline{P}$. Formally, let

$$\overline{P} = \bigcup_{r\in\mathcal{R}} (\mathcal{P}_r \setminus \{p_r\}) \subset \bigcup_{r\in\mathcal{R}} \mathcal{P}_r. \tag{77}$$

Intuitively, given a path assignment P, the complementary path assignment $\overline{P}$ consists of all the paths *not* taken. We can see the routing optimization constraints as a slotted assignment problem akin to the caching problem we have studied so far in the following way. Each request $r \in \mathcal{R}$ is associated with exactly $|\mathcal{P}_r| - 1$ slots. These slots are to be occupied by paths *not taken.* That is, each such slot is to be occupied by a path p in $\mathcal{P}_r$; whenever such a path p is stored in a slot, it is added in the complementary path assignment $\overline{P}$. We denote by $\mathcal{D}'$ the set of feasible complementary path assignments under this setting, that is:[7]

$$\mathcal{D}' = \left\{ \overline{P} \subset \prod_{r\in\mathcal{R}} \mathcal{P}_r : |\overline{P} \cap \mathcal{P}_r| \leq |\mathcal{P}_r| - 1 \right\}. \tag{78}$$

Then, given a content allocation A and complementary path assignment $\overline{P}$, the cost of serving a request $r = (i, b)$ is:

$$C_r(A, \overline{P}) = \sum_{p\in\mathcal{P}_r\setminus\overline{P}} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(A \cap \left\{ \bigcup_{k'\in[k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} = \emptyset \right). \tag{79}$$

[7] To better cast this as an assignment problem, we would need to introduce notation for slots per request, but this is equivalent to Eq. (78).

Similarly, the caching gain of a request $r = (i, b)$ due to caching at intermediate nodes and path assignment is:

$$\begin{aligned} f_r(A, \overline{P}) &= C_r(\emptyset, \emptyset) - C_r(A, \overline{P}) \\ &= \sum_{p\in\mathcal{P}_r} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(p \in \overline{P} \vee A \cap \left\{ \bigcup_{k'\in[k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} \neq \emptyset \right). \end{aligned} \tag{80}$$

The caching gain maximization problem amounts to:

$$\underset{A,\overline{P}}{\text{maximize}} \quad f(A, \overline{P}) = \sum_{t=1}^{T} f^t(A, \overline{P}) = \sum_{t=1}^{T} \sum_{r\in\mathcal{R}^t} f_r(A, \overline{P}), \tag{81a}$$

$$\text{subject to} \quad A \in \mathcal{D}, \overline{P} \in \mathcal{D}'. \tag{81b}$$

which is a submodular maximization over an (assignment) partition matroid w.r.t. both A and $\overline{P}$ (complementary set of P). The assingment nature of the matroid follows from the representation of complementary paths as slots "taken"; we prove submodularity below:

Lemma H.1. *Function $f : \mathcal{S} \times \mathcal{C} \times \prod_{r\in\mathcal{R}} \mathcal{P}_r \to \mathbb{R}_+$ is monotone and submodular w.r.t. both A and $\overline{P}$.*

Proof. In this proof, we switch to vector notation rather than the set notation. For any content allocation $A \in \mathcal{S} \times \mathcal{C}$, the corresponding allocation vector is denoted by $\boldsymbol{x} = [\mathbb{1}_{\{(s,i)\in A\}}]_{(s,i)\in\mathcal{S}\times\mathcal{C}} = [x_{(s,i)}]_{(s,i)\in\mathcal{S}\times\mathcal{C}}$. For any complementary path assignment $P \in \prod_{r\in\mathcal{R}} \mathcal{P}_r$, the corresponding complementary path assignment $\bar{P}$ is represented by vector $\bar{\boldsymbol{h}} = [\mathbb{1}_{\{(r,p)\notin P\}}]_{(r,p)\in\prod_{r\in\mathcal{R}}\mathcal{P}_r} = [\bar{h}_{r,p}]_{(r,p)\in\prod_{r\in\mathcal{R}}\mathcal{P}_r}$. The caching gain of a request (i, b) is:

$$f_{(i,b)}(\bar{\boldsymbol{h}}, \boldsymbol{x}) = \sum_{p\in\mathcal{P}_{(i,b)}} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \left(1 - (1 - \bar{h}_{(i,b),p}) \prod_{k'=1}^{k} (1 - x_{p_{k'}i}) \right), \tag{82}$$

which has the same form as Eq. (5). By Lemma 3.1, f is also monotone (non-decreasing) and submodular w.r.t. both $\boldsymbol{x}$ and $\bar{\boldsymbol{h}}$. The feasible set for $\bar{\boldsymbol{h}}$ is:

$$\sum_{p\in\mathcal{P}_r} \bar{h}_{r,p} = |\mathcal{P}_r| - 1, \forall r \in \mathcal{R}, \tag{83}$$

which is also a partition matroid. □

The monotonicity of the objective implies that an optimal solution exists in which all routing slots of the complementary path assignment are taken, so that indeed only one path is truly selected.[8]

This submodular maximization assignment problem can be tackled by the modified DistributedTGOnline with $1 - 1/e$ guarantee through Cor. 4.2 as follows. Following [43], for each request (i, b), source node b maintains an extra slot $s = (b, 0)$ determining its path assignment. Correspondingly, the information needed to compute the reward is from all possible paths due to Eq. (80), besides its own path. The second step in the online algorithm is, thus, modified as:

- When a request (i, b) is generated, (rather than one additional control message is generated to collect and transmit information,) $|\mathcal{P}_{i,b}|$ additional control messages are generated, one over one path $p \in \mathcal{P}_{i,b}$ to collect and transmit information.

[8] Note that if an optimal solution contains fewer occupied slots, one with higher caching gain can be constructucted by adding more paths.

Note that the communication cost increases linearly with $|\mathcal{P}_{i,b}|$. Nevertheless, it is possible that randomization approaches akin to the ones used in [43], can lower this dependency with a corresponding increase in regret; exploring this is beyond our scope.

I ANYTIME REGRET GUARANTEE

Under the *doubling trick* [16], the algorithm proceeds in phases. In the first phase, it sets its (short-term) horizon to a time-window $W_0 = 1$. Whenever a phase ends (i.e., the short-term horizon expires), the algorithm resets its state, and doubles the time window, so that the short term horizon at phase $n + 1$ satisfies:

$$W_{n+1} = 2W_n, \quad \text{for all } n \in \{0, 1, \dots, k\}.$$

For the sake of notational simplicity, assume that $T = 2^{k+1} - 1$ for some $k \in \mathbb{N}$. By Theorem 4.1, DistributedTGOnline has bounded regret $c\sqrt{W_n}$ at the end of each short-term horizon W_n, where c is a constant independent of W_n. Thus,

$$\sum_{n=0}^{k} c\sqrt{W_n} = c\sum_{n=0}^{k} 2^{0.5n} = c\frac{2^{0.5(k+1)} - 1}{\sqrt{2} - 1} = c\frac{2^{0.5\log_2(T+1)}}{\sqrt{2} - 1} = c\frac{\sqrt{T + 1}}{\sqrt{2} - 1} \leq \frac{\sqrt{2}}{\sqrt{2} - 1}c\sqrt{T}. \tag{84}$$

In other words, using this doubling trick, we can obtain an anytime regret bound for any algorithm designed for a fixed time horizon, while worsening the bound by a constant factor (namely, $\frac{\sqrt{2}}{\sqrt{2}-1}$).

The same argument can be used to show that the update cost is sublinear when taking update costs in to account. In particular, the modified policy resets its state at most $k - 1$ times and k is logarithmic in T, thereby contributing at most an $O(\log T)$ term to the overall update cost.

Received August 2021; revised October 2021; accepted November 2021